Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

RGen: Generador de datos para benchmarking de cargas de trabajo Big Data

Estudante: Rubén Pérez Jove
Dirección: Roberto Rey Expósito
Juan Touriño Domínguez

A Coruña, septiembre de 2020.

A mi madre

Agradecimientos

En primer lugar quiero agradacerles a mis padres, mi hermano y mi familia cercana por quererme, cuidarme y permitirme siempre elegir el camino para llegar a ser lo que soy. A los amigos que llevan conmigo toda la vida, y a los que se cruzaron más tarde, por ser parte fundamental de mi personalidad. A aquellos compañeros de facultad que se convirtieron en amigos ya inseparables, y a la gran familia que he forjado durante mi Erasmus, por ser protagonistas sin duda de los mejores años de mi vida. Agradecer también a mis directores de proyecto por brindarme la oportunidad y depositar en mí la confianza para realizar este trabajo. Sin todos vosotros no habría sido posible, gracias.

Resumen

El presente Trabajo Fin de Grado (TFG) presenta el diseño e implementación de RGen, un generador de datos paralelo para el benchmarking de cargas de trabajo Big Data. La herramienta está desarrollada en Java bajo el paradigma de programación MapReduce, más concretamente haciendo uso del framework de procesamiento Apache Hadoop. Además, RGen soporta la generación de datos directamente sobre el sistema de ficheros distribuido de Hadoop, piedra angular del almacenamiento de los frameworks Big Data para procesamiento por lotes (batch processing). RGen conjuga una doble labor de integración de características preexistentes y desarrollo de nuevas funcionalidades en una herramienta independiente. El objetivo final que se persigue es la creación de una herramienta completa, paralela y escalable que reúna las funcionalidades necesarias, sin tener que depender de software de terceros, para la generación de datos de las distintas cargas de trabajo soportadas en la suite de benchmarking Big Data Evaluator (BDEv).

Las principales funcionalidades desarrolladas en este TFG son la generación de texto y grafos que cumplen las características definidas por las 4 Vs del Big Data: Volumen, Variedad, Velocidad y Veracidad. Se pone especial énfasis en esta última ya que en muchos benchmarks específicos la necesidad de una gran cantidad de información verídica es primordial. Para ello se ha escogido el modelo LDA, utilizado para la extracción de tópicos o temas tratados en una serie de documentos, para la generación de texto. Por otro lado, en cuanto a la generación de grafos se refiere, se realiza a partir del modelo Kronecker.

Para el desarrollo de RGen se han empleado prácticas bien asentadas en la Ingeniería del Software. En cuanto al diseño, se ha hecho uso de patrones de diseño y arquitecturales con el objetivo de conseguir una herramienta fácilmente mantenible y extensible, a la vez que se proporciona un código limpio y de calidad. Para facilitar la organización en el trabajo se ha utilizado Scrum, marco de desarrollo ágil basado en Sprints.

Con respecto a la evaluación del rendimiento y escalabilidad del generador de datos se ha realizado la experimentación tanto en un entorno local como en un clúster de altas prestaciones. Para ello se han evaluado distintas configuraciones tanto en el número de nodos como en la cantidad de datos a generar en paralelo.

La herramienta desarrollada se encuentra disponible para su descarga en el siguiente repositorio Git: `https://github.com/rubenperez98/RGen`.

Abstract

This BSc Thesis presents the design and implementation of RGen, a parallel data generator for benchmarking Big Data workloads. The tool is developed in Java under the MapReduce programming paradigm, more specifically making use of the Apache Hadoop processing framework. In addition, RGen supports the generation of data directly on the Hadoop distributed file system, cornerstone of the storage of Big Data frameworks for batch processing. RGen brings together a twofold task of integrating existing features and developing new functionalities in a standalone tool. The main objective is the creation of a complete, parallel and scalable tool that gathers the necessary functionalities without having to depend on third-party software to generate data for the different workloads supported by the Big Data Evaluator (BDEv) benchmarking suite.

The main functionalities developed in this BSc Thesis are the generation of text and graphs that meet the characteristics defined by the 4 Vs of Big Data: Volume, Variety, Velocity and Veracity. Special emphasis is placed on the last one since many specific benchmarks require a huge amount of truthful information. On the one hand, the LDA model has been used for text generation, which is employed for the extraction of topics or themes covered in a series of documents. On the other hand, graphs generation is based on the Kronecker model.

RGen has been developed following well-established practices in software engineering. Design and architectural patterns have been used with the aim of obtaining an easily maintainable and extensible tool, while also providing clean and quality code. Scrum, an agile development framework based on Sprints, has been used to facilitate work organization.

Regarding the performance evaluation and scalability of the data generator, multiple experiments have been carried out both in a local environment and in a high-performance cluster. Different configurations have been evaluated both in the number of nodes and the amount of data to be generated in parallel.

The developed tool is publicly available to download at the following Git repository: https://github.com/rubenperez98/RGen.

Palabras clave:

- Generador de datos
- MapReduce
- HDFS
- Apache Hadoop
- Java
- Big Data
- Benchmarking

Keywords:

- Data generator
- MapReduce
- HDFS
- Apache Hadoop
- Java
- Big Data
- Benchmarking

Índice general

1 Introducción **1**

1.1 Motivación . . . 1
1.2 Objetivos . . . 2
1.3 Trabajo relacionado . . . 3
1.4 Estructura de la memoria . . . 4

2 Conceptos previos **7**

2.1 Generadores de datos . . . 7
2.1.1 Modelo de las 4 Vs (V^4) . . . 7
2.1.2 Tipos de generadores . . . 8
2.2 Big Data . . . 10
2.2.1 MapReduce . . . 10
2.3 Modelos utilizados . . . 12
2.3.1 LDA . . . 12
2.3.2 Kronecker . . . 14

3 Tecnologías y herramientas **17**

3.1 Tecnologías Big Data . . . 17
3.1.1 Apache Hadoop . . . 17
3.1.2 HDFS . . . 18
3.1.3 Apache YARN . . . 20
3.1.4 BDEv . . . 21
3.2 Herramientas de desarrollo . . . 22
3.2.1 Java . . . 22
3.2.2 Git . . . 22
3.2.3 GitHub . . . 23
3.2.4 Eclipse . . . 23
3.2.5 Maven . . . 23

3.2.6 Vagrant 23

4 Metodología y plan de trabajo **25**
4.1 Scrum 25
4.1.1 Roles Scrum 25
4.1.2 Eventos Scrum 26
4.1.3 Aplicación a la forma de trabajo 27
4.2 GitFlow 28
4.2.1 Aplicación a la forma de trabajo 28

5 Funcionalidades **31**
5.1 Generación de texto: modelo LDA 31
5.2 Generación de grafos: modelo Kronecker 32

6 Diseño y desarrollo **35**
6.1 Preparación y estudio del dominio 35
6.2 Primer Sprint - Análisis del generador de HiBench 36
6.3 Segundo Sprint - Integración 39
6.4 Tercer Sprint - Generación de texto mediante LDA 41
6.5 Cuarto Sprint - Generación de grafos mediante Kronecker 43
6.6 Estimaciones y coste 44

7 Evaluación del rendimiento **47**
7.1 Entorno de pruebas 47
7.2 Diseño de las pruebas 48
7.3 Análisis de los resultados 50

8 Conclusiones y trabajo futuro **55**
8.1 Conclusiones 55
8.2 Trabajo futuro 56

Lista de acrónimos **59**

Glosario **61**

Bibliografía **63**

Índice de figuras

2.1 Esquema MapReduce . . . 11
2.2 Tópicos LDA . . . 13
2.3 Producto Kronecker de dos matrices A y B . . . 15
2.4 Fractal con patrones visibles en una matriz de adyacencia de un grafo Kronecker 15

3.1 Ecosistema Hadoop . . . 18
3.2 Arquitectura y funcionamiento de HDFS . . . 19
3.3 Replicación de bloques en HDFS . . . 20
3.4 Arquitectura YARN . . . 21

4.1 Esquema Scrum . . . 27
4.2 Esquema GitFlow . . . 29

6.1 Diagrama UML del generador DataGen de HiBench . . . 38
6.2 Planificación y estimación del Sprint 1 . . . 38
6.3 Diagrama UML del generador RandomTextWriter . . . 40
6.4 Diagrama UML del generador TeraGen . . . 40
6.5 Diagrama UML del generador KMeans . . . 41
6.6 Planificación y estimación del Sprint 2 . . . 41
6.7 Diagrama UML del generador LDA . . . 42
6.8 Planificación y estimación del Sprint 3 . . . 43
6.9 Diagrama UML del generador Kronecker . . . 44
6.10 Planificación y estimación del Sprint 4 . . . 44
6.11 Resumen de la planificación y estimación económica del desarrollo de RGen . 45
6.12 Resumen del coste y esfuerzo globales del proyecto . . . 45
6.13 Diagrama de Gantt del proyecto . . . 46

7.1 Escalabilidad débil (generación de texto) . . . 51

7.2 Escalabilidad fuerte (generación de texto) 51
7.3 Escalabilidad débil (generación de grafos) 52
7.4 Escalabilidad fuerte (generación de grafos) 52

Índice de tablas

1.1 Resumen de las técnicas de generación de datos utilizadas en los principales benchmarks Big Data 4

6.1 Cargas de trabajo y generadores de datos utilizados en BDEv 36

7.1 Pruebas de generación de texto 49
7.2 Pruebas de generación de grafos 49
7.3 Tiempos de ejecución para la generación de texto (en segundos) 50
7.4 Tiempos de ejecución para la generación de grafos (en segundos) 50

Capítulo 1
Introducción

A lo largo de este primer capítulo se ofrece una breve contextualización del presente Trabajo Fin de Grado (TFG), así como la división en capítulos escogida a la hora de redactar este documento. Lo primero en describir son los motivos fundamentales que nos llevan al desarrollo de RGen, seguido de las metas perseguidas a lo largo de su implementación, y finalizando con la presentación del estado del arte actual en relación a la generación de datos.

1.1 Motivación

En la actualidad, la cantidad de datos que se almacenan y procesan en los sistemas informáticos es gigantesca, y además dicha cantidad crece exponencialmente cada poco tiempo. Hoy en día cada persona está generando datos continuamente, ya sea a través del smartphone, tableta, ordenador portátil, wearables, televisores, etc. Todos estos datos necesitan ser procesados para convertirlos en conocimiento útil con el que poder tomar decisiones y mejorar la experiencia del consumidor. Sin embargo, las tecnologías y técnicas tradicionales de procesamiento no siempre resultan adecuadas, ya que pueden no ser suficientes para afrontar semejante cantidad de datos y/o hacerlo en un tiempo razonable. Es por ello que en este contexto surge un conjunto de tecnologías específicas enfocadas al procesamiento de volúmenes de datos masivos, campo conocido como Big Data.

Una de las definiciones más extendidas de Big Data es el conjunto de técnicas, tecnologías y herramientas a través de las cuales es posible almacenar, procesar, analizar y visualizar volúmenes de datos que debido a su gran tamaño no son posibles de tratar con los métodos convencionales. Para ello, se basa en el almacenamiento y procesado distribuido de los datos sobre los nodos de un clúster que por sí solos no podrían afrontar esta carga de trabajo. Para evaluar el rendimiento de los frameworks de procesamiento, existen herramientas o suites de benchmarking que se encargan de generar determinadas cargas de trabajo con el objetivo de obtener diferentes métricas como tiempo de CPU, memoria consumida, acceso a disco, etc.

Una de las problemáticas que surgen alrededor de estas herramientas es la necesidad de disponer de conjuntos de datos de entrada de naturaleza y volumen específicos que nutran las cargas de trabajo a evaluar. Estos datos pueden ser extraídos de fuentes preexistentes, con todo el trabajo de preparación que conlleva y sus múltiples limitaciones, o generados sintéticamente. En relación a los generadores de datos, no existen en la actualidad herramientas que engloben todas las características y funcionalidades idóneas en una única solución. La mayoría de suites de benchmarking existentes utilizan diversas soluciones en la fase de generación de los datos de entrada. De esta carencia surge la motivación principal de este proyecto que es el desarrollo de un generador de datos, RGen, basado en el paradigma MapReduce [1] y que englobe las funcionalidades necesarias para nutrir cualquier tipo de carga de trabajo.

1.2 Objetivos

Como se ha mencionado en la sección anterior, el objetivo principal es el desarrollo de un generador de datos que aúne en una única solución todas las funcionalidades que se esperan de una herramienta como esta. Estas funcionalidades pasan por la generación de cualquier volumen de datos de cualquier tipo (texto, grafos, etc.) de forma escalable y en un tiempo razonable, además de garantizar cierto grado de veracidad en los datos generados. En resumen, significa cumplir los requisitos asociados a lo que se conoce como las 4 Vs del Big Data: Volumen, Variedad, Velocidad y Veracidad. Más en concreto, el conjunto de funcionalidades específicas que debería reunir RGen viene dado por las distintas cargas de trabajo Big Data que soporta la suite de benchmarking Big Data Evaluator (BDEv) [2, 3], en la cual se integrará el generador desarrollado en este TFG. Esta suite utiliza para algunas cargas de trabajo el generador de otra suite de benchmarking, HiBench [4], además de algunas soluciones propias de Hadoop, entre otras. Cabe destacar además que todas estas opciones de generación producen su salida directamente en el sistema de ficheros distribuido de Hadoop, Hadoop Distributed File System (HDFS) [5], y siguiendo el paradigma de programación paralela MapReduce [1]. Además del proceso de integración de todas estas soluciones en una única herramienta, otro de los objetivos marcados en este proyecto es el desarrollo de nuevas formas de generar algún tipo de dato en concreto, teniendo en cuenta los requisitos de las 4 Vs. Se ha escogido la generación de texto a partir del modelo Latent Dirichlet Allocation (LDA) [6, 7] y la generación de grafos mediante el modelo Kronecker [8, 9]. Por último, como objetivo heredado del anterior, se busca la evaluación de las nuevas funcionalidades desarrolladas en un entorno clúster de altas prestaciones.

Resumiendo, los objetivos de este proyecto son:

- Desarrollar un nuevo generador de datos, RGen, basado en el paradigma MapReduce, con las siguientes características:

  - Integración en una única herramienta de soluciones existentes utilizadas por benchmarks Big Data representativos.
  - Soporte para la generación de datos en paralelo y directamente en el sistema de ficheros distribuido HDFS.
  - Desarrollo de nuevas funcionalidades de generación de datos de una naturaleza concreta y que tengan en cuenta los requisitos de las 4 Vs, especialmente la Veracidad.

- Evaluar experimentalmente el rendimiento de RGen en un entorno clúster de altas prestaciones real, utilizando diferentes configuraciones y número de nodos.

1.3 Trabajo relacionado

Existen diferentes suites de benchmarking y generadores de datos empleados en cada una de ellas (ver resumen en Tabla 1.1). A continuación se describe brevemente el estado del arte en este aspecto, citando aquellos trabajos previos más estrechamente relacionados con RGen. Algunas herramientas de benchmarking como HiBench [4] incorporan un generador de datos propio (DataGen), mientras que BigDataBench [10] integra Big Data Generator Suite (BDGS) [11]. Otros, en cambio, se nutren de varias soluciones para la fase de generación, como es el caso ya mencionado de BDEv, que utiliza el generador DataGen de HiBench junto con otras soluciones independientes, como clases nativas de Hadoop u otras librerías. Existen también generadores totalmente independientes como Parallel Data Generation Framework (PDGF) [12], pero no soporta la generación de datos directamente sobre HDFS.

Por un lado, RGen unifica todas las funcionalidades del generador de HiBench implementadas en Hadoop. Estas funcionalidades se definen internamente como el tipo de carga de trabajo a la que alimentan. Por lo tanto, las características del generador de HiBench que se preservan en RGen citadas por su nombre interno son: PageRank, Nutch, Hive y Bayes. Los datos que generan son, por lo tanto, característicos y específicos de las cargas de trabajo que les dan nombre. Por otro lado, RGen también integra las funcionalidades de generación de texto RandomTextWriter y de generación de bytes TeraGen propias de Hadoop, realizando las modificaciones oportunas. Por último, se integra también la clase GenKMeansDataset proveniente del paquete de clústering Mahout[13], utilizada en BDEv para la ejecución de la carga de trabajo denominada KMeans.

Además de la integración de soluciones existentes, RGen proporciona dos nuevas funcionalidades para la generación basada en conjuntos de datos reales: 1) generación de texto a partir del modelo LDA, y 2) generación de grafos con el modelo Kronecker. Ambas funcionalidades se inspiran en BigDataBench, cuyo generador BDGS no soporta HDFS y por tanto su salida se almacena en un sistema de ficheros local. RGen reimplementa ambos algoritmos

Categorías de técnicas	Herramientas de benchmarking	Ventajas principales	Limitaciones
Conjuntos de datos preexistentes	MRBS, Streams, DSIMBench, GenBase, Stream Bench, PUMA, HcBench, CloudSuite	Preserva por completo la veracidad de los datos	Control escaso o nulo sobre el volumen de datos a generar y gran coste en la transferencia de los datos al clúster Hadoop por generarse en un sistema de ficheros local
Generación basada en distribuciones sintéticas	Hadoop micro benchmarks, TestDFSIO, TPC-x-HS, BG, TPC-C, TPC-H, PigMix, HiBench, YCSB, WGB, Graphalytics, BigBench, TPC-DI	El volumen de datos es escalable y se tiene gran control en la velocidad de generación	No garantiza la veracidad de los datos generados
Generación basada en datos reales	LinkBench, BigDataBench	El volumen de datos es escalable y se tiene gran control en la velocidad de generación, además de un cierto control en la veracidad de los datos generados	Complejidad añadida por la necesidad de modelar los datos, algunos tipos de datos son difíciles de modelar y es complicado evaluar su veracidad
Híbridos	CALDA, AMPLab benchmark, TPC-E, TPC, DS, GraphBIG		

Tabla 1.1: Resumen de las técnicas de generación de datos utilizadas en los principales benchmarks Big Data

usando el paradigma MapReduce para soportar la generación en paralelo y además poder almacenar los datos de salida directamente en HDFS.

1.4 Estructura de la memoria

A continuación se muestra una breve descripción del contenido de cada capítulo en el que se estructura el presente documento:

1. **Introducción:** En este primer capítulo se brinda una descripción general del contexto que enmarca el proyecto desarrollado en este TFG y sus objetivos, así como los diversos aspectos tratados a lo largo de la memoria.

2. **Conceptos previos:** En el segundo capítulo se exponen los fundamentos teóricos sobre los que descansa RGen, descripción de términos relacionados con los generadores de

datos actuales, conceptos de Big Data y modelos de inteligencia artificial aplicados en la generación de texto y grafos.

3. **Tecnologías y herramientas:** El tercer capítulo presenta de forma resumida las tecnologías subyacentes empleadas en el desarrollo de RGen, tanto aquellas que son específicas del ámbito Big Data en el que se enmarca este TFG, como otras más generales que están presentes en la mayoría de proyectos de desarrollo software.

4. **Metodología y plan de trabajo:** La aplicación de la metodología ágil Scrum, utilizada para aumentar la productividad durante todo el ciclo de vida del proyecto, se describe en este cuarto capítulo.

5. **Funcionalidades:** En el quinto capítulo se describen las nuevas funcionalidades que proporciona RGen, la generación de texto a partir de un corpus dado con el modelo LDA y la generación de grafos sintéticos con el modelo Kronecker.

6. **Diseño y desarrollo:** El sexto capítulo describe cada una de las partes o Sprints en las que se ha dividido el proyecto para su desarrollo. Además, se presenta una estimación de los costes económicos asociados al proyecto.

7. **Evaluación del rendimiento:** El séptimo capítulo presenta las pruebas experimentales realizadas con RGen desplegado en un entorno clúster de altas prestaciones. En concreto, se evalúa la escalabilidad fuerte y débil de las nuevas funcionalidades proporcionadas por RGen.

8. **Conclusiones y trabajo futuro:** Por último, se exponen las principales conclusiones del TFG y posibles líneas de trabajo futuro relacionadas con el desarrollo de RGen.

Capítulo 2

Conceptos previos

A lo largo de este capítulo se exponen los fundamentos teóricos sobre los que se basa la herramienta desarrollada, necesarios para poder entender su funcionamiento, capacidades y limitaciones.

2.1 Generadores de datos

La generación de los datos de entrada es una fase clave en las herramientas de benchmarking para Big Data, ya que es necesario generar y transformar datos para cargas de trabajo específicas de una forma escalable y en un tiempo razonable. La generación u obtención de la entrada necesaria para la ejecución de una carga de trabajo es normalmente la primera fase de cualquier prueba de esfuerzo sobre una determinada infraestructura. Sin una cantidad de datos adecuada y de una determinada naturaleza, es imposible realizar pruebas de evaluación que realmente caractericen el rendimiento de una infraestructura o framework de procesamiento. Una posibilidad sería utilizar como entrada conjuntos de datos reales. Sin embargo, obtener el volumen de datos deseado con el formato y naturaleza adecuados para determinadas pruebas puede volverse imposible. Necesitamos además que esta generación se realice en un tiempo razonable, ya que no es deseable establecer en esta primera fase un cuello de botella sobre la duración total de los experimentos.

2.1.1 Modelo de las 4 Vs (V^4)

Para cumplir los requisitos expuestos anteriormente, buscando una generación de datos idónea, la literatura recomienda cumplir las famosas 4 Vs del Big Data [14]: Volumen, Velocidad, Variedad y Veracidad.

Volumen: La característica fundamental que marca la generación de datos es la cantidad a generar en una escala de grandes volúmenes de datos. En Big Data se suele trabajar con escalas de GB (GigaByte), TB (TeraByte), e incluso PB (PetaByte), EB (ExaByte) y ZB (Zet-

taByte). Además, para algunos tipos de datos que se describen a continuación (e.g. los grafos), es habitual tomar como unidad de medida el número total de nodos del grafo a generar. Cada año aumenta considerablemente el número de datos generados por persona, debido principalmente a la cantidad de dispositivos Internet of Things (IoT) que están saliendo al mercado. Para proporcionar un contexto, la consultora IDC predice que los dispositivos IoT generarán aproximadamente 80 ZB de datos en 2025 [1]. Es por ello que la generación de datos tiene que evolucionar acorde con el volumen que genera la sociedad, para que las evaluaciones de rendimiento de las infraestructuras en las que se van a procesar sean significativas.

Velocidad: La siguiente característica importante a tener en cuenta es la velocidad a la que se generan los datos. Como en el punto anterior, es importante adaptar la velocidad a la que se generan los datos a la escala a la que lo hace la sociedad, ya que van a ser esas infraestructuras las que tengan que procesar toda esa cantidad de información. Se puede poner como ejemplo el caso de las redes sociales como Twitter, donde se están generando aproximadamente 21 millones de tweets cada hora durante este año 2020, según la consultora de marketing digital David Sayce [2].

Variedad: La variedad denota la diversidad en los tipos, estructuras y fuentes de los datos a generar. Toda esta variedad de datos puede ser clasificada en tres tipos: 1) datos no estructurados (e.g. texto, imágenes, audio, vídeo), 2) datos semi-estructurados (e.g. grafos, XML), y 3) datos estructurados (e.g. tablas relacionales).

Veracidad: Probablemente la menos obvia de las cuatro, pero con una gran importancia dependiendo de la carga de trabajo que se vaya a realizar con los datos generados. La veracidad de los datos se refiere a la naturaleza de los mismos en cuanto al nivel de confianza, corrección o semejanza con datos del mismo tipo existentes en el mundo real. Para ello muchas veces es necesario preprocesar los datos que se han obtenido de forma experimental, o adoptar formas de generar datos que sigan modelos reales, como los desarrollados en este proyecto.

2.1.2 Tipos de generadores

Dependiendo de la forma en la que los datos sean generados, podemos clasificar los distintos tipos de generadores en cuatro grandes grupos (mencionados en la Tabla 1.1 del capítulo anterior) que se describen a continuación [15].

Conjuntos de datos preexistentes: Realmente no es un generador *per se*, ya que se proporcionan como entrada conjuntos de datos ya existentes con un determinado tamaño fijo y que están preprocesados y listos para su consumo. Muchas herramientas de benchmarking proporcionan esta opción ya que permite eliminar por completo el tiempo correspondiente al proceso de generación de datos. En este caso podría utilizarse cualquier conjunto de datos real,

[1] https://www.idc.com/getdoc.jsp?containerId=prUS45213219
[2] https://www.dsayce.com/social-media/tweets-day/

como por ejemplo entradas de la Wikipedia, grafos de conexiones de amistades de Facebook, etc. Además, también existe la posibilidad de proveer diferentes conjuntos de datos de un tamaño variable y ajustable a diversas necesidades. Sin embargo, esta opción limita bastante la posibilidad de trabajar con un volumen de datos específico, además de que la descarga de los datos puede ser muy lenta.

Basados en distribuciones sintéticas: En esta categoría podemos distinguir entre los que generan datos de forma aleatoria siguiendo una distribución uniforme por defecto y los que siguen una distribución no uniforme. Entre los primeros encontramos, por ejemplo, clases propias del framework Hadoop como RandomText y RandomTextWriter. Estas son utilizadas comúnmente por cargas de trabajo como WordCount, Sort, Grep, etc. Un ejemplo del segundo caso podemos encontrarlo en PDGF [12]. Este método genera datos estructurados en forma de tabla relacional utilizando tanto generación de números aleatorios a partir de una semilla como siguiendo una distribución matemática como la normal, Poisson, exponencial, etc. La veracidad de estos métodos, sin embargo, depende en gran medida de los parámetros utilizados. Otro ejemplo perteneciente a esta categoría sería DataGen, el generador interno de HiBench [4], dado que basa todas sus funcionalidades en la implementación de una distribución Zipfian [16].

Basados en datos reales: Este tipo de generadores capturan parámetros de conjuntos de datos reales y veraces, preservando así sus características principales y escalando la muestra al tamaño deseado. El mejor ejemplo de este grupo lo encontramos en el generador BDGS proporcionado por BigDataBench [11]. Este generador cuenta con 15 conjuntos de datos reales, representativos y preprocesados, de los tres tipos de datos mencionados anteriormente. A partir de dichos datos reales, BDGS es capaz de producir conjuntos de datos del tamaño deseado preservando su características principales. BDGS se toma como ejemplo a la hora de desarrollar RGen, paralelizando y adaptando su funcionamiento al paradigma MapReduce y generando los datos directamente en HDFS. Además, en RGen se implementan dos nuevas funcionalidades catalogadas dentro de este grupo de generadores. La primera es la generación de texto a partir del modelo LDA, y la segunda es la generación de grafos a partir del modelo Kronecker.

Híbridos: Por último existe también la posibilidad de que un generador reúna varias de las características descritas anteriormente, intentando de esta forma cubrir todo el espectro posible. El generador desarrollado en este proyecto, RGen, pertenece a este grupo de generadores ya que reúne funcionalidades pertenecientes a los grupos anteriormente descritos.

RGen

Después de exponer los tipos de generadores que nos podemos encontrar, y las características básicas que deben cumplir, cabe justificar las decisiones de diseño que se van a tomar

en este proyecto para satisfacer estos requisitos. Como se ha comentado, RGen formaría parte del grupo de generadores híbridos ya que integra funcionalidades pertenecientes a más de un grupo. Por una parte, adopta algunas de las características de DataGen, así como las clases RandomTextWriter y TeraGen de Hadoop, pertenecientes al grupo de generadores basados en distribuciones sintéticas. Por otra parte, la generación de texto (LDA) y de grafos (Kronecker) se basan en datos preexistentes y reales para crear nuevos conjuntos de datos, por lo que se englobarían en el grupo de generadores basados en datos reales.

RGen cumple además con las 4 Vs en lo que a generación de datos se refiere:

- Volumen: Permite especificar el tamaño deseado de los datos a generar en todas sus funcionalidades, pudiendo escalar este factor dependiendo del número de *mappers* especificado para Hadoop. Además, hay que tener en cuenta que la generación se realiza directamente sobre un sistema de ficheros distribuido, lo que permite mayor capacidad y recursos.
- Velocidad: Al tratarse de una herramienta paralela, implementando el paradigma MapReduce, el tiempo que se necesita para generar un conjunto de datos de un tamaño concreto puede ajustarse dependiendo de los recursos asignados a la tarea. Por lo tanto, la velocidad de la herramienta puede variarse según los recursos utilizados.
- Variedad: Es posible generar diferentes tipos de datos: texto, grafos, tablas, etc.
- Veracidad: Las funcionalidades de generación de texto LDA y de grafos Kronecker nos permiten conseguir un alto grado de veracidad, ya que basan su funcionamiento en datos de entrada reales, obteniendo un conjunto natural con unas características similares.

2.2 Big Data

En el Capítulo 1 se ha definido el término Big Data como el conjunto de técnicas, tecnologías y herramientas a través de las cuales es posible almacenar, procesar y analizar volúmenes de datos que debido a su gran tamaño no es posible hacerlo con los métodos convencionales. Debido al crecimiento de Internet, uso de las redes sociales y la aparición de toda clase de dispositivos IoT, el Big Data se encuentra entre las tecnologías más en auge en este momento. Y es que sin él no sería posible tratar la cantidad de datos que los sistemas manejan actualmente.

Esta sección se centra en el paradigma MapReduce de Google y tecnologías relacionadas, por ser la base teórica subyacente del funcionamiento de RGen.

2.2.1 MapReduce

MapReduce [1] es un paradigma de programación paralelo que se basa en dividir los datos de entrada en bloques para distribuirlos entre los nodos de un clúster y procesarlos mediante

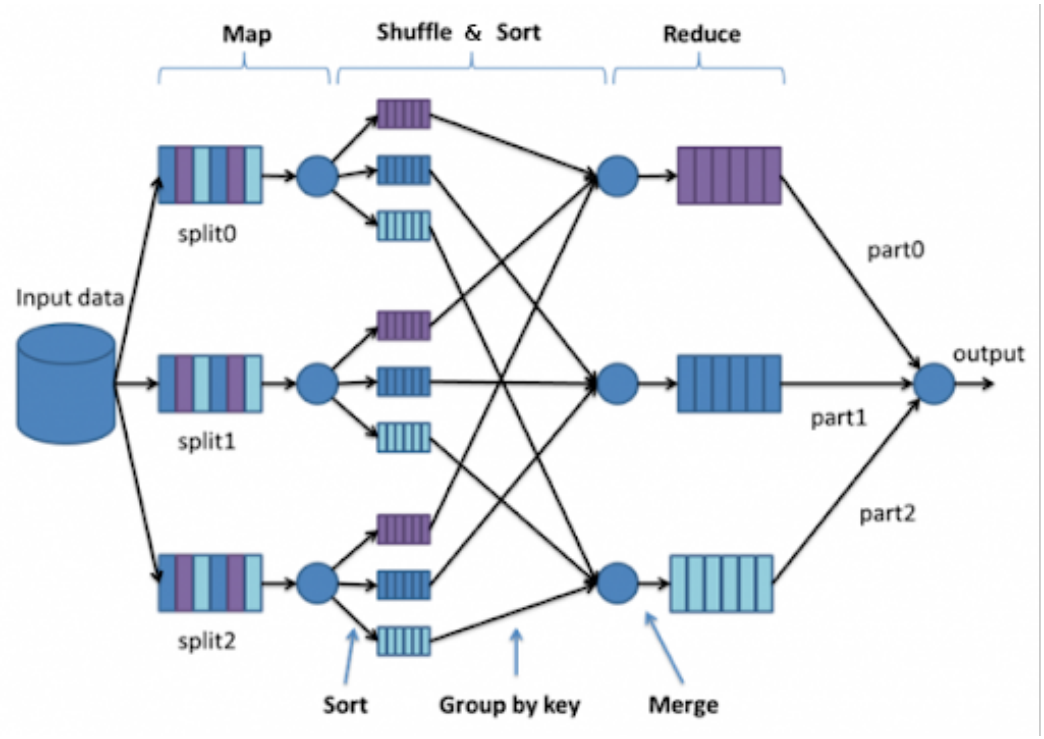


Figura 2.1: Esquema MapReduce

las dos fases que nombran al modelo. Este modelo fue inicialmente diseñado y publicado por Google, contando con un framework propietario en C++ que lo implementa. Asociado al modelo MapReduce, Google publicó también una implementación propietaria de un sistema de ficheros distribuido denominado Google File System (GFS) [17], basado en una arquitectura maestro/esclavo.

Las bases de este paradigma son la posibilidad de desarrollar aplicaciones paralelas escritas a alto nivel, la interacción mínima entre los nodos del clúster y la distribución de datos inherente y tolerante a fallos. En esencia, el proceso completo cuenta con dos fases programables (véase Figura 2.1): *Map* y *Reduce*. Además, en la transición entre ambas fases existe un proceso de *Shuffle and Sort* de los datos intermedios. Cada una de estas fases puede contar con múltiples tareas, distribuidas entre los nodos del clúster.

En la fase *Map* se ejecuta el algoritmo descrito por el programador sobre una porción discreta de los datos de entrada. Esta entrada está formada por el procesamiento de datos en pares clave-valor. Una vez realizadas las operaciones sobre estos pares, la salida completa de la fase *Map* estará formada por una lista de cero o más pares clave-valor.

$$\text{MAP} (<\text{K\_IN} , \text{V\_IN}>) \text{ -> LIST} (<\text{K'} , \text{V'}>)$$

En la fase *Reduce* se toman como entrada todos los valores con la misma clave que se han

generado en la fase *Map*, para producir pares clave-valor de salida de acuerdo al algoritmo de reducción descrito por el desarrollador. Por lo tanto, los datos intermedios entre la fase *Map* y *Reduce* son agrupados y ordenados por clave, por lo que el tipo de dato de la clave debe ser comparable:

REDUCE (<K' , LIST (V')>) -> LIST (<K_OUT , V_OUT>)

La fase intermedia entre *Map* y *Reduce* (*Shuffle and Sort*) es la encargada de agrupar todos los valores de salida de la fase *Map* de una misma clave y ordenarlos. Además también es posible realizar la fase *Shuffle and Sort* para la salida de cada tarea *Map*, ahorrando tiempo de procesamiento en la fase *Shuffle and Sort* general.

2.3 Modelos utilizados

En esta sección se presentan los modelos de inteligencia artificial seleccionados en este TFG para extraer los parámetros necesarios para la generación de texto y grafos sintéticos a partir de conjuntos de datos reales, preservando sus características principales y garantizando así la veracidad de los datos. Mediante la implementación de estos modelos se han desarrollado dos funcionalidades nuevas en RGen, además de integrar otras soluciones existentes. No es parte del alcance de este proyecto, sin embargo, la utilización de estos modelos para la modelización de nuevos conjuntos de datos reales, sino la generación de datos a partir de ellos.

La elección de estos modelos se basa en la capacidad que han demostrado para extraer las características principales de un conjunto de datos real, en nuestro caso de un conjunto de documentos y de un grafo, para posteriormente generar la cantidad de datos deseada conservando sus características intrínsecas.

2.3.1 LDA

El modelo Latent Dirichlet Allocation (LDA) [6, 7] es una de las formas más populares de modelado de tópicos. Esto se refiere a la tarea de identificar tópicos o temas de los que tratan un conjunto de documentos. Resumiendo su funcionamiento, LDA preestablece un número fijo de tópicos presentes en una serie de documentos. Cada tópico es representado por un conjunto de palabras que semánticamente están relacionadas. La labor de LDA es mapear todos los documentos en busca de estos tópicos en la forma en la que las palabras de un documento son mayormente capturadas por esos tópicos preestablecidos.

Para ilustrar el concepto se expone el siguiente ejemplo: imaginemos un conjunto formado por 1,000 documentos con aproximadamente 1,000 palabras cada uno de ellos, donde además cada documento tiene una media de 500 palabras que se repiten entre ellos. Una posible forma

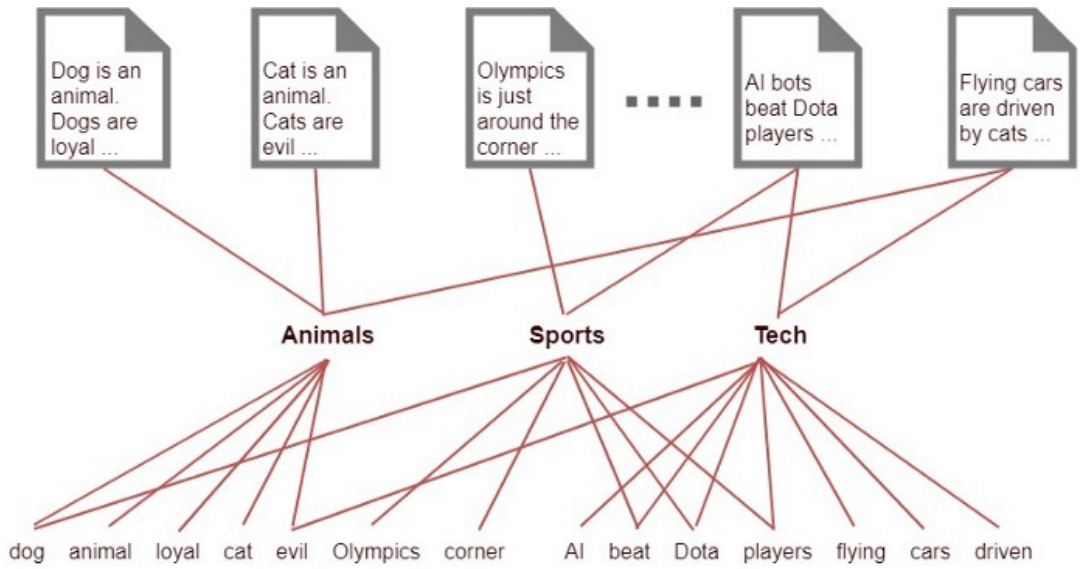


Figura 2.2: Tópicos LDA

de conocer de qué trata cada documento sería establecer un enlace entre los documentos y las palabras que se repiten, teniendo en cuenta el número de ocurrencias de las mismas. Este procedimiento, sin embargo, nos daría una cantidad aproximada de 500,000 enlaces para el ejemplo propuesto, lo cual no es manejable ni escalable. Es por ello que se realiza una aproximación que estima el número de tópicos que se repiten a lo largo de los documentos (e.g. 10 tópicos). El modelo conecta las palabras a los tópicos dependiendo de cuán preciso es su significado de acuerdo a un tópico determinado, para después representar un documento como un conjunto de tópicos dependiendo de las palabras que forman el documento y el propio tópico. La Figura 2.2 muestra este ejemplo de una forma visual, aunque el nombre de los tópicos son representativos puesto que no se conocen. Estos vienen dados en realidad por una relación de palabras y factores, como por ejemplo: Animales = (0.3*Perros, 0.4*Gatos, 0.2*Leal, 0.1*Feroz).

Esta idea posee detrás un complejo fundamento matemático basado en métodos de inferencia variacional y que extraen información acerca de los tópicos que son tratados en un conjunto de documentos, basándose en conjuntos de palabras asociados a un valor matemático. Una vez elaborado el modelo para un conjunto de datos real concreto (e.g. entradas de la Wikipedia), la generación de texto sintético puede realizarse a partir de la creación de un diccionario con las palabras de los documentos y la generación de documentos nuevos que mantengan estos tópicos de forma inherente. Como se ha mencionado anteriormente, la creación de los modelos no forma parte de los objetivos de este proyecto, por lo que para la generación de texto se toman como fuente modelos previamente creados y extraídos del generador BDGS. Sin embargo, en caso de querer obtener un nuevo modelo LDA que sirva de

parámetro de entrada para RGen y a partir del cual se pueda generar texto sintético, puede utilizarse alguna de las herramientas que hay disponibles públicamente, como por ejemplo Mr. LDA [18] (versión paralela implementada en Hadoop) o lda-c [19] (versión secuencial implementada en C).

2.3.2 Kronecker

El modelo Kronecker [8, 9] es un proceso de modelado de grafos a partir del cual se pueden extraer distintos parámetros característicos de un conjunto de nodos y vértices y generar así un grafo que preserve su naturaleza. Este modelo es especialmente útil a la hora de generar grafos sintéticos que obedezcan a patrones que representen la realidad, también llamados grafos naturales. Existe una gran cantidad de fenómenos pertenecientes al mundo real como conexiones de personas, páginas web o rutas, que pueden ser representados en los sistemas informáticos a través de grafos. Por este motivo son especialmente útiles para almacenar información y procesarla con el objetivo de generar nuevo conocimiento a partir de ellos, como modelos predictivos o de comportamiento. En lo que respecta a RGen, es crucial contar con métodos para generar este tipo de datos además de la importancia de poder producir la cantidad deseada en un tiempo razonable y de una naturaleza dada, y no simples conexiones aleatorias entre nodos y vértices que no tengan significado alguno. Además, la obtención de grandes cantidades de información volcadas en grafos naturales es una tarea complicada y tediosa. El modelo Kronecker nos permite generar grafos sintéticos naturales con características similares a las de los grafos reales.

Lo primero es definir qué propiedades cumplen los grafos naturales. Aunque no es una definición fácil de acotar, la mayoría de grafos naturales cumplen con las siguientes características principales:

- Distribución en ley de potencias del grado de los nodos: Un número pequeño de nodos tienen una gran cantidad de conexiones entre ellos (grado), mientras que un grafo con un gran número de nodos suele tener pocas conexiones entre ellos.
- Auto-similitud: En los grafos naturales, las conexiones a gran escala entre partes del grafo suelen reflejar las conexiones a pequeña escala entre sus nodos. Esta propiedad es la que siguen los fractales.

El núcleo del modelo Kronecker reside en una simple operación entre matrices llamada producto Kronecker. Definiendo A como una matriz $m \times n$ y B como otra matriz $m' \times n'$, su producto Kronecker se muestra en la Figura 2.3. La idea es que la matriz B se multiplica repetidamente por cada elemento de la matriz A. Se puede extrapolar esta idea a los grafos, los cuales pueden ser representados por matrices a través de la matriz de adyacencia, en la cual las filas y columnas representan los nodos y la información de las celdas representa las

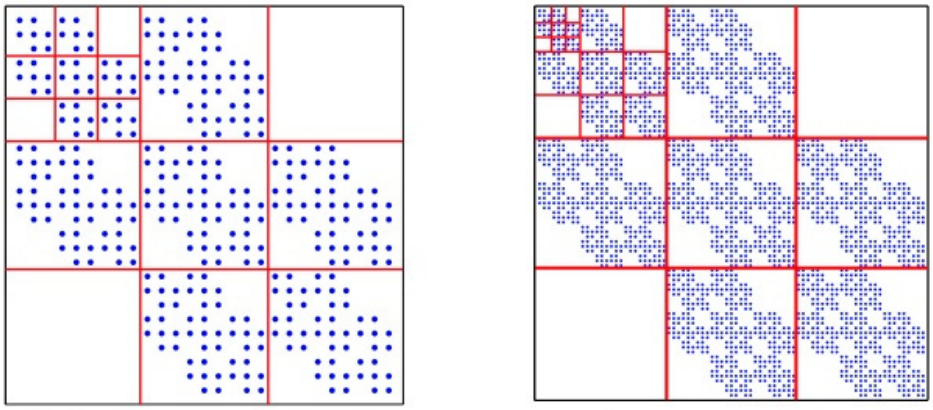
$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

Figura 2.3: Producto Kronecker de dos matrices A y B

aristas (1 ó 0 dependiendo de si existe conexión entre dos nodos). Se puede definir, por tanto, el producto Kronecker de dos grafos como el producto Kronecker de sus matrices de adyacencia. El resultado sería un nuevo grafo con la estructura principal del primer grafo y donde cada nodo es una copia del segundo grafo.

Con la idea principal de multiplicar un grafo por otro, surge el grafo Kronecker, el cual se define como el producto Kronecker de un grafo por sí mismo. Además, si realizamos esta misma operación múltiples veces, obtenemos la herramienta fundamental de la generación de grafos naturales y auto-similares: las potencias Kronecker. Si a un grafo dado le aplicamos una potencia Kronecker de grado K, obtenemos el producto del grafo inicial por sí mismo K veces. En la Figura 2.4 puede apreciarse la similitud entre escalas de un grafo Kronecker.

Por último, solo necesitaremos un grafo generador que represente de alguna forma a un grafo real, a partir del cual poder generar grafos sintéticos del tamaño deseado y preservando las características del original. La creación de este grafo generador escapa a los objetivos de este proyecto, por lo que de nuevo utilizaremos modelos ya diseñados de grafos reales previamente generados. Estos grafos generadores han sido extraídos nuevamente del generador BDGS, el cual cuenta con una serie de modelos previamente analizados en base a grafos reales. Como en el caso de LDA, cualquier modelo con las características adecuadas puede utilizarse



(a) Matriz de adyacencia K_3 (27x27) (b) Matriz de adyacencia K_4 (81x81)

Figura 2.4: Fractal con patrones visibles en una matriz de adyacencia de un grafo Kronecker

como parámetro para nuestro generador, además de poder utilizar los modelos extraídos de BDGS. Para obtener la matriz generadora a partir de una red real se puede utilizar cualquier herramienta que implemente el algoritmo *KronFit* [8], como por ejemplo la librería de análisis y minado de grafos denominada Stanford Network Analysis Platform (SNAP) [20].

Capítulo 3

Tecnologías y herramientas

En este capítulo se describen brevemente las tecnologías utilizadas para llevar a cabo el desarrollo de la herramienta RGen. Entre ellas se hace distinción entre implementaciones específicas de paradigmas teóricos explicados en el capítulo anterior y herramientas de desarrollo y despliegue de aplicaciones, entre otras.

3.1 Tecnologías Big Data

Para comenzar se hace un repaso de las tecnologías orientadas a tratar grandes cantidades de datos, las cuales se denominan Big Data. Debido a la naturaleza de la herramienta a implementar, se ha escogido un framework de procesamiento distribuido especializado en dicho ámbito: Apache Hadoop.

3.1.1 Apache Hadoop

Apache Hadoop [21] es un framework de código abierto y escalable orientado al almacenamiento, procesamiento y análisis de grandes volúmenes de datos que está desarrollado completamente en Java. El primer componente fundamental de Hadoop es su motor de procesamiento de datos basado en el paradigma MapReduce de Google (véase Sección 2.2.1). El segundo componente principal de Hadoop es el sistema de ficheros distribuido HDFS [5], inspirado a su vez en la implementación GFS de Google. Básicamente, HDFS divide la información en bloques de tamaño fijo que son replicados por los distintos nodos del clúster para proporcionar así tolerancia a fallos. Su funcionamiento y arquitectura se detalla en la Sección 3.1.2. Existe un tercer componente que desde la versión 2 de Hadoop es otra pieza fundamental: el planificador y gestor de recursos Yet Another Resource Negotiator (YARN) [22], el cual se detalla en la Sección 3.1.3. Además del núcleo central, Hadoop cuenta con un ecosistema muy amplio de proyectos de código abierto (e.g. Hive, Pig) que le nutren de funcionalidades adicionales (ver Figura 3.1).

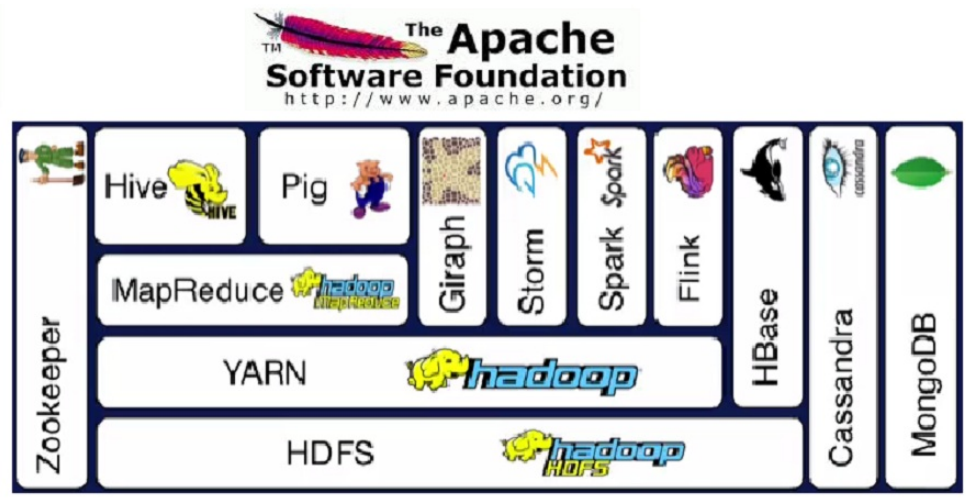


Figura 3.1: Ecosistema Hadoop

El despliegue de un clúster Hadoop puede realizarse de tres modos: local, pseudo-distribuido y distribuido. El modo local es la configuración por defecto una vez instalado Hadoop, donde se ejecuta todo el entorno en un único proceso Java. Es el modo más útil para desarrollo y pruebas y es el que se ha utilizado a lo largo del desarrollo de RGen. En el modo pseudo-distribuido, Hadoop se ejecuta en un único nodo pero con sus múltiples componentes corriendo en procesos Java distintos, simulando un clúster real. Por último, en el modo distribuido se instala Hadoop en cada nodo de un clúster y se realizan las configuraciones pertinentes para establecer la conectividad entre ellos. El modo distribuido se utiliza en entornos de producción o en entornos de experimentación como la evaluación de rendimiento de RGen llevada a cabo en este TFG (Capítulo 7).

3.1.2 HDFS

Hadoop Distributed File System (HDFS) [5] es un sistema de ficheros distribuido diseñado para ejecutarse sobre hardware commodity y optimizado para el almacenamiento y escritura/-lectura secuencial de grandes ficheros. La principal diferencia frente a otros sistemas de ficheros distribuidos es su alto grado de tolerancia a fallos, proporcionando redundancia entre los nodos que lo conforman, evitando que si uno de ellos falla los datos se pierdan. Proporciona además una interfaz que facilita la abstracción de su funcionamiento interno. Sin embargo, resulta de interés conocer su arquitectura y funcionamiento a bajo nivel para entender las operaciones que realiza y así poder aprovechar mejor sus ventajas.

El primero de los conceptos clave de su funcionamiento es el almacenamiento de los datos en bloques como unidad mínima de lectura y escritura, con un tamaño configurable por el usuario (por defecto, 128 MB). Además, pueden almacenarse los distintos bloques de un

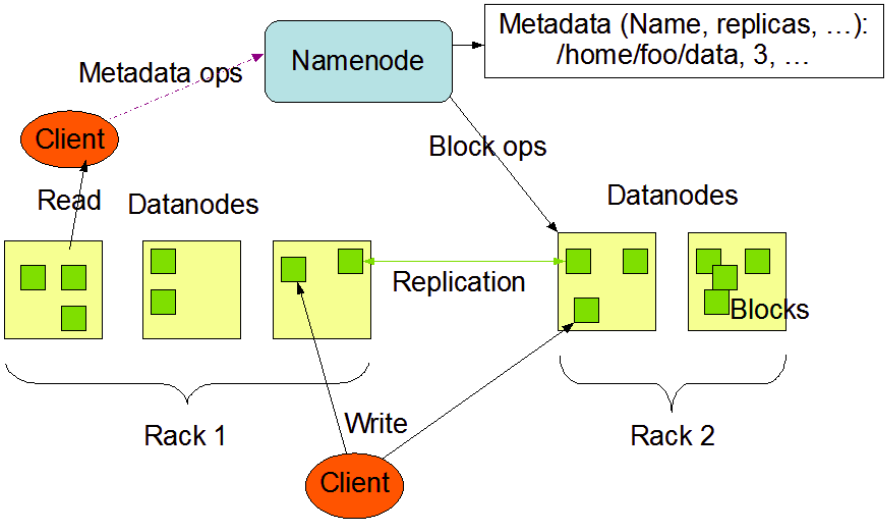


Figura 3.2: Arquitectura y funcionamiento de HDFS

determinado fichero de forma distribuida entre los nodos que conforman el clúster Hadoop, lo que permite la lectura paralela de los bloques del fichero y la posibilidad de trabajar con ficheros más grandes que la capacidad de almacenamiento de los nodos por separado.

Para manejar la información de a qué fichero corresponde y dónde se encuentra almacenado cada bloque entre los distintos nodos del clúster, HDFS distingue dos tipos de servicios: *Namenode* y *Datanode* (ver Figura 3.2). Un *Namenode* actúa como ”master”, almacenando los metadatos necesarios para construir el sistema de ficheros sobre los *Datanodes*. Estos metadatos contienen información como la estructura de directorios y ficheros, el nodo donde se encuentra un bloque de datos determinado o una lista con la ubicación de los distintos bloques que forman un fichero. Además, mantiene balanceado el número de bloques en cada nodo y, si un *Datanode* falla, es capaz de detectarlo rápidamente y replicar los bloques perdidos entre los nodos restantes (ver Figura 3.3). Por otra parte, un *Datanode* puede considerarse como un esclavo que se limita a almacenar los bloques que el *Namenode* le solicita.

El proceso de lectura de un fichero por parte de un cliente consiste en solicitar al *Namenode* la información relativa a dicho fichero. El *Namenode* le envía al cliente una tabla de metadatos localizando los *Datanodes* donde se encuentran los bloques que forman el fichero. El cliente usará esta información para solicitar a cada uno de los *Datanodes* los datos correspondientes. En cuanto al proceso de escritura, este comienza con la solicitud de escritura por parte del cliente al *Namenode*. Este realiza unas comprobaciones previas, como espacio disponible y permisos, y cuando da el visto bueno, el cliente particiona el fichero en bloques, y cada uno de ellos serán escritos de forma secuencial en múltiples *Datanodes*. Para cada bloque el cliente solicita al primer *Datanode* que escriba un bloque en disco. Una vez realizado, el

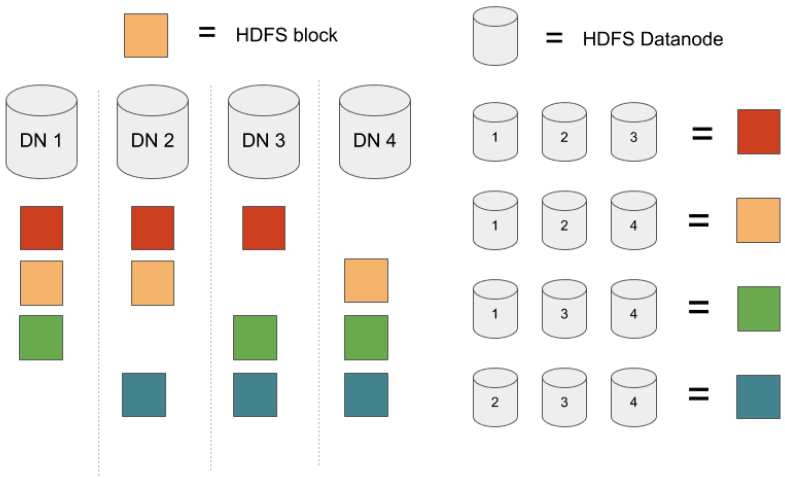


Figura 3.3: Replicación de bloques en HDFS

propio *Datanode* se encargará de replicar el bloque al siguiente *Datanode* de forma transparente para el cliente. Una vez que esta cadena termina el cliente recibe un ACK y, una vez que todos los bloques del fichero se encuentren almacenados, el cliente manda una confirmación al *Namenode* que da por finalizado el proceso de escritura.

3.1.3 Apache YARN

Apache Yet Another Resource Negotiator (YARN) [22, 23] proporciona un planificador agnóstico a los trabajos que se encuentran en ejecución en un clúster. YARN separa, por una parte, la gestión de recursos, y por otra, la planificación y monitorización de trabajos, dividiendo en servicios distintos ambas gestiones. Para ello cuenta con tres componentes que se muestran en la Figura 3.4:

- *ResourceManager*: Se encarga de arbitrar los recursos entre todas las aplicaciones del clúster. Cuenta con dos subcomponentes: el *Scheduler* y el *ApplicationsManager*. El *Scheduler* es el responsable de asegurar los recursos necesarios para las aplicaciones que están en ejecución, mientras que el *ApplicationsManager* es el encargado de aceptar las peticiones de trabajos y gestionar los contenedores en los que se ejecutan los *ApplicationMaster*.

- *NodeManager*: Es un agente que se encuentra corriendo en todos los nodos esclavo del clúster y es el responsable de mantener y monitorizar sus propios recursos, para reportarlos al *ResourceManager*.

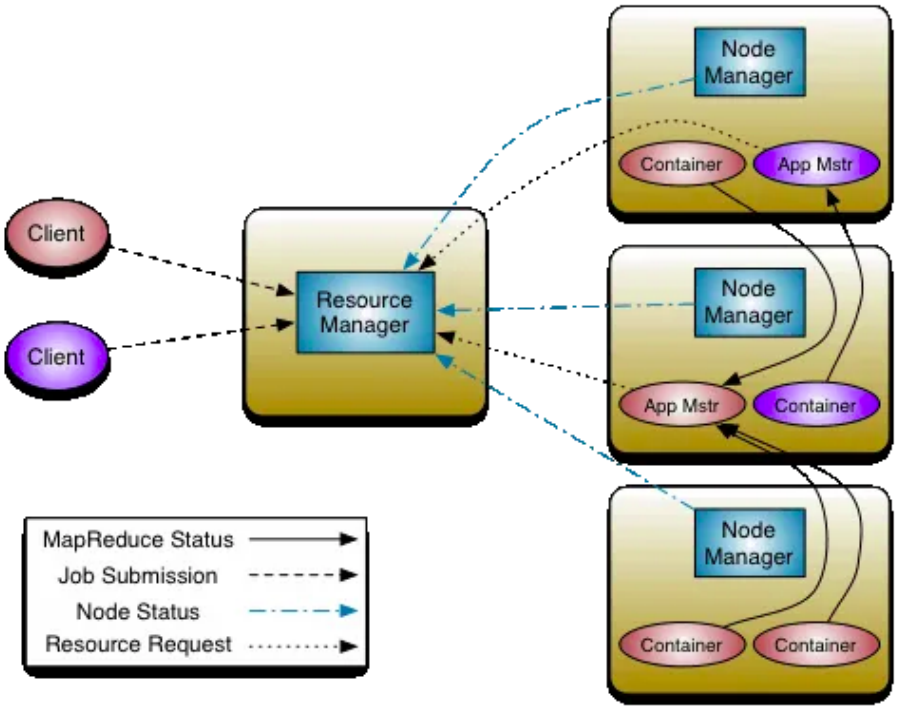


Figura 3.4: Arquitectura YARN

- *ApplicationMaster*: Es el componente encargado de coordinar la ejecución de una determinada aplicación en el clúster.

3.1.4 BDEv

Big Data Evaluator (BDEv) [2, 3] es una herramienta desarrollada en el Grupo de Arquitectura de Computadores (GAC) de la Universidade da Coruña que permite el despliegue y evaluación de diferentes frameworks de procesamiento Big Data en términos de rendimiento, utilización de recursos, eficiencia energética y eventos a nivel de microarquitectura. BDEv agiliza el proceso de despliegue en un clúster de diferentes entornos para los mútiples frameworks que soporta, entre los cuales se encuentran Hadoop, Flame-MR, Spark, Flink y DataMPI, entre otros. Además, soporta una amplia lista de benchmarks específicos dependiendo del framework a utilizar y del tipo de datos a procesar. Algunos de los benchmarks soportados por BDEv son: WordCount, Sort, TeraSort, Grep, Bayes, PageRank y KMeans.

RGen surge en parte de la necesidad de este tipo de herramientas de benchmarking de unificar en una sola solución independiente la generación de los conjuntos de datos de entrada para alimentar a los diversos benchmarks disponibles. Es por ello que se integran en RGen

algunas de las soluciones usadas actualmente en BDEv para la generación de datos, además de añadir nuevas funcionalidades.

BDEv es especialmente útil a la hora de desplegar automáticamente distintos entornos de prueba en un clúster, realizando la configuración de los distintos componentes (e.g. HDFS, YARN) de una forma estándar y sin necesidad de desplegar manualmente el entorno para cada prueba. También se encarga de la configuración de los recursos hardware (i.e. CPU, memoria, disco, red) y se integra de forma transparente con el gestor de recursos disponible en el clúster para obtener automáticamente los nodos asignados a cada experimento.

3.2 Herramientas de desarrollo

En esta sección se describe y justifica el uso de las herramientas que fueron necesarias o de especial utilidad para el desarrollo del proyecto.

3.2.1 Java

Java es un lenguaje de programación de propósito general, rápido, seguro, fiable y orientado a objetos. Es compilado a un lenguaje intermedio (bytecode), el cual es interpretado por una implementación de la máquina virtual de Java o Java Virtual Machine (JVM). Por lo tanto es independiente de la plataforma y sistema operativo que se utilice, aislando así la ejecución del programa y gestionando la memoria de una forma automática y eficiente. Una de sus principales ventajas es el catálogo de librerías disponibles (sistema de ficheros, fecha y hora, matemáticas, etc.). Es por ello que cuenta con un gran nivel de abstracción y limpieza en el código.

Se ha escogido este lenguaje de programación para llevar a cabo el desarrollo de RGen ya que es la principal API de Hadoop disponible para realizar la implementación con el paradigma MapReduce. Además, se contaba con una amplia experiencia en el uso del mismo.

3.2.2 Git

Git es un sistema de control de versiones distribuido y descentralizado que permite el desarrollo de código en diversas ramas o branches, uso de etiquetas, logs, etc. Actualmente es el sistema de control de versiones más utilizado en todo el mundo. Entre sus principales ventajas se encuentra el uso de punteros para la implementación de las ramas, lo que supone un gran ahorro de espacio en el repositorio. Además permite trabajar sin necesidad de disponer de un servidor central, utilizando repositorios locales.

3.2.3 GitHub

GitHub es el repositorio público utilizado para almacenar el código desarrollado en este TFG, utilizando el sistema de control de versiones Git. Se ha escogido este repositorio por la amplia aceptación que tiene en la comunidad y por las funcionalidades que nos brinda, además de la facilidad de uso de su interfaz.

3.2.4 Eclipse

Eclipse es un entorno de desarrollo integrado o Integrated Development Environment (IDE) multiplataforma y desarrollado en Java, con soporte para múltiples lenguajes y altamente personalizable por el usuario. Está catalogado como el IDE por excelencia entre los desarrolladores Java. Entre sus múltiples ventajas se encuentra la opción de añadir plugins para obtener funcionalidades extra, como puede ser el uso de Maven o Git de forma integrada. Se ha escogido este IDE debido a la experiencia en su uso y su gran acoplamiento con el resto de herramientas utilizadas.

3.2.5 Maven

Maven es un gestor de construcción de proyectos que permite automatizar la gestión de dependencias, empaquetado, despliegue del proyecto, etc. Maven guarda la configuración de cada proyecto en un archivo XML en la raíz del mismo (pom.xml). En él se definen diversos parámetros como la versión del proyecto, el nombre, las dependencias, etc. También permite añadir plugins para ampliar sus funcionalidades en situaciones específicas.

3.2.6 Vagrant

Vagrant es una herramienta para la creación y gestión de máquinas virtuales mediante un flujo de trabajo simple. Vagrant es compatible con la mayoría de software de virtualización del mercado, entre los más destacados VirtualBox o VMware. Enfocado en la automatización, Vagrant disminuye los tiempos de instalación y arranque de las máquinas virtuales, permitiendo una mayor productividad. Esta herramienta se ha utilizado para el despliegue de un entorno de pruebas en el que se instaló Hadoop en modo local de una forma automática y sencilla. Tanto el Vagrantfile con la configuración de la máquina virtual, como los ficheros necesarios para su instalación, se encuentran disponibles en el repositorio GitHub de RGen.

Capítulo 4

Metodología y plan de trabajo

En este capítulo se describe la metodología y forma de trabajo escogidas a la hora de desarrollar el presente proyecto, así como la adaptación de algunas características a la naturaleza específica del mismo. Se ha decidido utilizar la metodología ágil Scrum con el objetivo de llevar a cabo un desarrollo incremental basado en Sprints. Además se optó por el flujo de trabajo GitFlow, el cual permite aislar las funcionalidades brindando un desarrollo fluido y evitando el acoplamiento de las mismas.

4.1 Scrum

Scrum [24] es un proceso que nos proporciona unas pautas para centrarnos en conseguir un producto con el mayor valor y en el menor tiempo posible, permitiendo inspeccionar el software real de trabajo rápidamente y en repetidas ocasiones. Para ello se forman equipos auto-organizados con el fin de determinar la prioridad de las tareas a entregar, posibilitando que en intervalos de tiempo fijo se pueda ver el software desarrollado funcionando y decidir si es posible liberarlo o seguir mejorándolo en una fase siguiente. Esta metodología ágil es empleada en la actualidad por empresas tan prestigiosas como Microsoft, Google y Yahoo para el desarrollo de proyectos de diferente clase y envergadura.

Las características principales de Scrum son la utilización de equipos auto-organizados y motivados donde prima la flexibilidad y la adaptación, el avance del producto en una serie de Sprints de duración fija (entre dos y cuatro semanas) y el agrupamiento de los requisitos en forma de elementos de una lista conocida como el *Product Backlog*.

4.1.1 Roles Scrum

En la metodología Scrum se distinguen distintos roles que cumplen diversas funciones en el ciclo de vida de un producto software. El primer rol es el de *Product Owner*. Este rol es el representante del cliente, el cual posee el producto y cuenta con la visión global del

resultado deseado. Por tanto, especifica los requisitos que debe cumplir finalmente el software a desarrollar y determina las prioridades entre los mismos.

El segundo rol es el de *Scrum Master*. Es un experto en la metodología Scrum que supervisa que todo el proceso se lleve a cabo siguiendo las pautas establecidas y las buenas prácticas de Scrum. Además, se encarga de eliminar impedimentos que puedan retrasar al equipo de desarrolladores, tomar decisiones rápidas que agilicen el proceso y, en definitiva, estrechar la cooperación entre todos los roles y funciones de Scrum asegurando un equipo funcional y productivo.

Por último, se encuentra el *Development Team*. Es un equipo auto-organizado que se encarga del desarrollo del producto. Está compuesto normalmente entre 5 y 9 miembros, con un tamaño ideal de 7 personas. Debe ser un grupo multifuncional de programadores, testers, analistas, diseñadores, etc. en los que no existe ninguna jerarquía de mando ni título significativo, a menos que así se decida internamente para favorecer la productividad. Solo puede haber cambios en los miembros de un equipo entre los Sprints. El *Development Team* debe disponer de un entorno de trabajo que favorezca la interacción y comunicación entre sus miembros.

4.1.2 Eventos Scrum

Los eventos en Scrum son definidos como reuniones de distinta duración y finalidad dentro del proceso general. Estas reuniones están diseñadas para favorecer la comunicación entre los roles, agilizando cualquier problemática que pudiera surgir a lo largo del procedimiento.

El primer evento que tiene lugar antes de la ejecución de cada Sprint es la reunión de planificación. Aquí se reúnen los tres roles que conforman el escenario Scrum para elaborar un documento llamado *Product Backlog*. Este consta de una lista de requisitos que debe cumplir el producto final, ordenados por una prioridad establecida por el *Product Owner*. Una vez obtenida esta lista, el *Development Team* elabora una segunda lista de tareas necesarias para hacer cumplir los requisitos dispuestos, ordenada también según las prioridades demandadas. A partir de esta segunda lista se elabora el *Sprint Backlog*, documento en el que se recogen las tareas a realizar durante el Sprint.

A partir de aquí comienza el Sprint, que puede tener una duración variable de entre 2 y 4 semanas, supervisado por el *Scrum Master*. En cada Sprint, al principio del día se realiza una reunión diaria que conforma el siguiente evento Scrum. Esta reunión breve se desarrolla entre los miembros del *Development Team* y el *Scrum Master*, la cual tiene una duración aproximada de 15 minutos y en la que cada miembro responde a estas tres preguntas:

- ¿Qué he hecho desde la última reunión?
- ¿Qué voy a hacer a partir de ahora?
- ¿Qué impedimentos tengo o puedo llegar a tener para realizar la tarea?

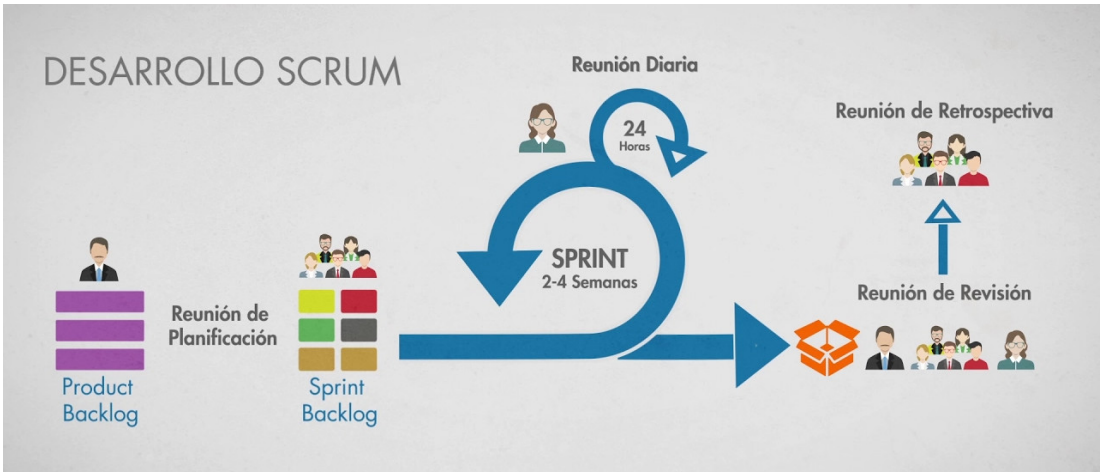


Figura 4.1: Esquema Scrum

Al finalizar cada Sprint tienen lugar dos eventos adicionales, la reunión de retrospectiva y la reunión de revisión. En la primera, los miembros del *Development Team* analizan el trabajo desarrollado a lo largo del Sprint, buscando los errores cometidos y las formas de mejorar la productividad de cara a Sprints futuros. La reunión de revisión, sin embargo, junta a todos los roles de nuevo para entregar los requisitos previstos al *Product Owner*, el cual verifica que cumplan las expectativas marcadas, adaptando y replanificando los requisitos restantes de cara al próximo Sprint. La Figura 4.1 muestra de forma visual el flujo de trabajo de Scrum junto con todos los eventos descritos previamente.

4.1.3 Aplicación a la forma de trabajo

Debido a la naturaleza de un TFG, se han adaptado diversas características de la metodología Scrum al escenario en el que se ha trabajado. La función del *Product Owner* ha sido desempeñada por ambos directores del proyecto, los cuales han especificado los requisitos que debería cumplir la herramienta. Tanto el *Development Team* como el *Scrum Master* han sido desempeñados por el autor del TFG, en cuanto a que es un proyecto individual.

Se ha dividido el trabajo en cuatro Sprints, los cuales se explicarán en detalle en el Capítulo 6. Cada Sprint tiene una duración de entre 4 y 6 semanas, extendiendo un poco la duración recomendada, con el fin de adaptarla a los plazos marcados. Al principio de cada Sprint se ha llevado a cabo una reunión de planificación, elaborando el correspondiente *Product Backlog* y *Sprint Backlog* con los requisitos y tareas deseados en cada funcionalidad. Cada día de trabajo se produce lo que sería una reunión diaria personal, en la que me respondo a mí mismo las tres preguntas mencionadas en la Sección 4.1.2. Por último, al final de cada Sprint se realiza el análogo adaptado a las reuniones de revisión y retrospectiva.

4.2 GitFlow

GitFlow es un flujo de trabajo que nos marca unas directrices, a la hora de gestionar un repositorio Git, con las que tener un mayor control y organización en el proceso de integración continua. Las ventajas que nos ofrece son el aumento en la velocidad de entrega de código, la disminución de errores humanos en la mezcla de ramas y la eliminación de dependencias de funcionalidades.

Se basa principalmente en dos ramas, la rama *develop* y la rama *master*. En la rama *develop* convergen las ramas de desarrollo, y la rama *master*, también conocida como rama principal o de producción, es donde residen las distintas versiones con código funcional desplegado en producción. A partir de la rama *master* se crean otras ramas como puede ser *hotfix/bug_id*, utilizadas para corrección de errores inesperados. Además de las mencionadas, la rama *develop* es el origen de las ramas *feature*, utilizadas para el desarrollo de nuevas funcionalidades, y *release*, utilizadas para preparar nuevas versiones a desplegar así como para realizar últimos ajustes de código. En la Figura 4.2 se muestra de manera visual el flujo de trabajo de un proyecto que utiliza GitFlow.

4.2.1 Aplicación a la forma de trabajo

Teniendo en cuenta las necesidades y características de un TFG, además de realizarse un desarrollo de forma individual, se ha optado por realizar pequeñas modificaciones sobre GitFlow a la hora de establecer un flujo de trabajo. Se ha eliminado la rama *develop*, integrándola en la rama *master*. Para cada funcionalidad se ha creado una rama *feature* nueva, como marca el modelo, pero también se han eliminado las ramas *release*. Por último, se ha creado una única rama *hotfix* para solventar problemas inesperados de ramas ya desarrolladas.

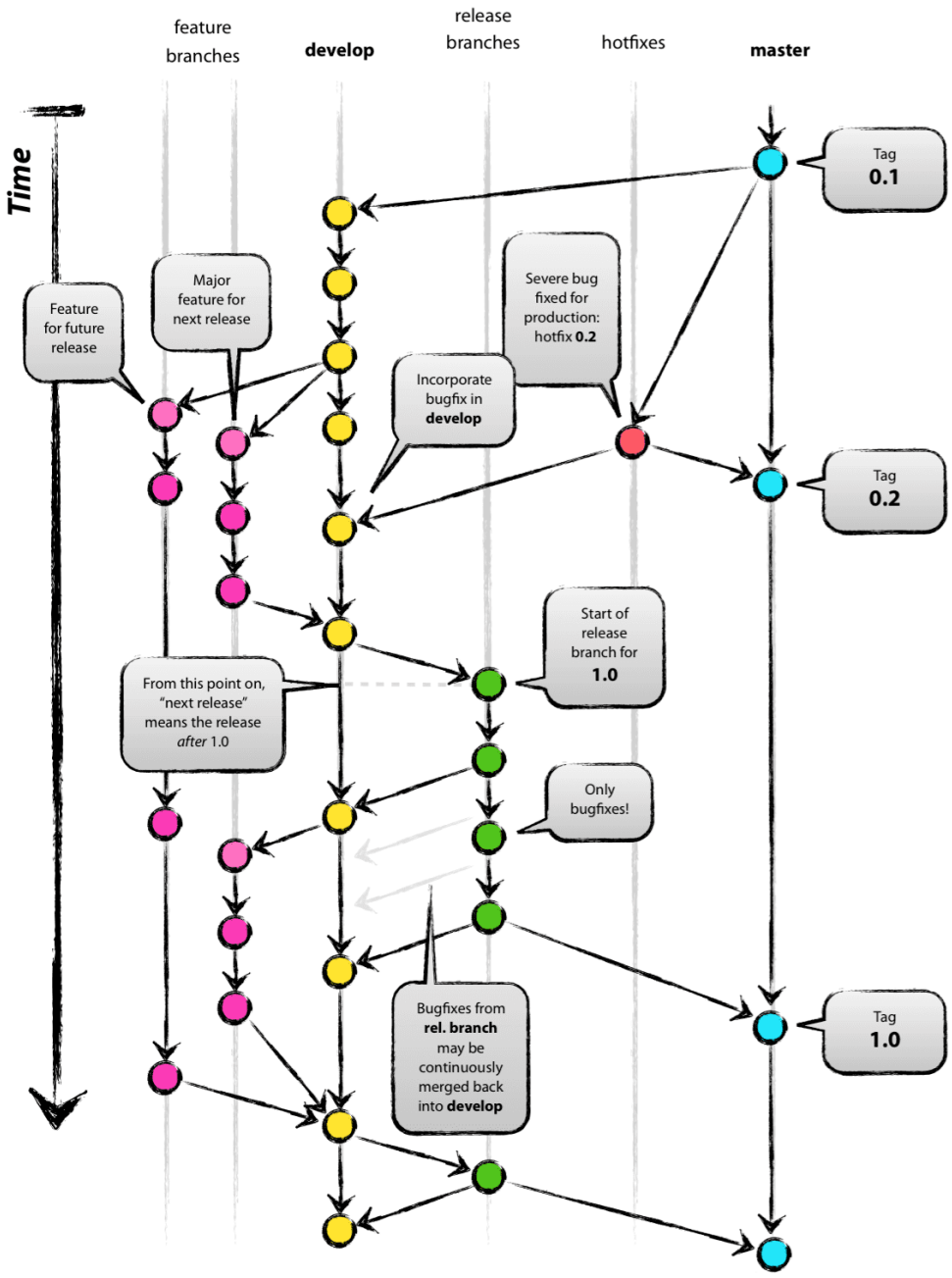


Figura 4.2: Esquema GitFlow

Capítulo 5

Funcionalidades

En este capítulo se exponen las nuevas funcionalidades que se han desarrollado específicamente en este TFG: la generación de texto a partir del modelo LDA y la generación de grafos utilizando el modelo Kronecker. Los fundamentos teóricos de ambos modelos se han explicado previamente en las Secciones 2.3.1 y 2.3.2, respectivamente, por lo que este capítulo se centra en exponer los servicios y características principales que nos brindan estas nuevas funcionalidades en RGen.

5.1 Generación de texto: modelo LDA

Esta funcionalidad nos permite, a partir del modelado LDA de un conjunto de documentos de entrada previamente analizados, generar texto realista que preserve las cualidades de los documentos originales (véase Sección 2.3.1 para una descripción más detallada). El texto generado se forma a partir de un diccionario de palabras que se extrae del conjunto de documentos que forman la entrada inicial. Además, plasma la distribución de tópicos heredada del modelo LDA.

Por lo tanto, el parámetro fundamental es el modelo LDA (directorio con la información correspondiente). Es posible también, a través de parámetros especificados por el usuario, controlar la cantidad de información a generar (i.e. bytes totales), el número de líneas por documento, el número medio de palabras por línea y, por último, el número de documentos a generar.

El algoritmo implementado para la generación de texto, el cual es una versión paralela usando MapReduce del algoritmo propuesto por el generador BDGS [11], sigue los siguientes puntos:

- Se escoge un modelo LDA previamente generado a partir de un conjunto de datos de entrada, como por ejemplo un conjunto determinado de entradas extraídas de la Wikipedia.

- Para cada documento a generar se realizan los siguientes pasos, tantas veces como líneas (o bytes) se deseen generar por documento:

    1. Para cada línea del documento se escoge una longitud de línea que sigue una distribución de Poisson con una media especificada por el usuario (parámetro "número de palabras por línea").
    2. Para cada palabra a generar de esa línea, se escoge un tópico que viene dado por un valor aleatorio que sigue una distribución multinomial de theta [1].
    3. La palabra escogida será aquella que venga dada por una distribución multinomial de beta [2], dependiente del tópico escogido previamente.

RGen integra tres modelos LDA (*lda_wiki1w*, *wiki_1w5* y *wiki_noSW_90_Sampling*) generados con distintos parámetros a partir de 4.3 millones de artículos en inglés extraídos de la Wikipedia. Estos modelos residen en un determinado directorio en el repositorio del proyecto (/src/main/resources), por lo que se pueden usar directamente sin necesidad de generar un nuevo modelo. Sin embargo, cualquier modelo puede ser utilizado para la generación de texto siempre que se sigan las siguientes directrices: 1) la información del modelo debe residir en un directorio con su nombre, y 2) este directorio debe contener tres ficheros: final.beta, final.other y <nombre_del_modelo>.voca. El contenido de cada uno de estos ficheros es el siguiente:

- final.beta: lista de valores que representan la probabilidad de que una palabra del diccionario aparezca en un tópico dado. El fichero debe contener tantas líneas como tópicos estimados por el modelo y, por cada línea, una lista de valores de longitud igual al número de palabras del diccionario.
- final.other: fichero resumen con tres líneas que indican, por orden: número de tópicos, número de términos (palabras) y parámetro alpha.
- <nombre_del_modelo>.voca: diccionario de palabras, una por cada línea.

5.2 Generación de grafos: modelo Kronecker

La generación mediante el modelo Kronecker nos permite generar grafos sintéticos que preserven la estructura y forma de grafos extraídos de la realidad, y por tanto que cuenten con un grado de veracidad y confianza elevados, más allá de la generación de nodos y aristas de forma aleatoria. Como se explicó en la Sección 2.3.2, el modelo Kronecker cumple estos

[1] Theta es un conjunto de valores aleatorios extraídos de una distribución Gamma usando como parámetros: alpha (obtenido del modelo LDA) y 1.

[2] Distribución de probabilidades de aparición de cada palabra del diccionario en un tópico dado.

requisitos permitiéndonos producir grafos con una distribución en ley de potencias del grado de los nodos y auto-similitud, ambos parámetros fundamentales en la generación de grafos naturales. Para ello se hace uso de una matriz inicial, normalmente de 2x2 o 3x3, con valores significativos propios del modelo. Tomando como operación fundamental las potencias Kronecker, se multiplica el tamaño de esta matriz de forma exponencial con base la dimensión de la matriz generadora, para así generar matrices de adyacencia de grafos del tamaño deseado.

Los parámetros fundamentales para este generador son: el número de iteraciones K que realizaremos sobre la matriz inicial, y la matriz en sí misma. El valor de K representa, por lo tanto, el exponente de la potencia Kronecker: el número de veces que aplicaremos el producto Kronecker sobre la matriz inicial. En nuestra implementación, el parámetro K viene dado por un número entero, y la matriz generadora por una representación lineal de la forma [value11,value12...;value21,value22...;...]. Además, contamos con parámetros adicionales como son el directorio de salida de los datos generados, el número de *mappers* y *reducers* del trabajo Hadoop a ejecutar y el delimitador utilizado para representar una arista como la conexión entre dos nodos del grafo.

El algoritmo para la generación de grafos, el cual de nuevo es una implementación paralela usando MapReduce del algoritmo propuesto por BDGS [11], sigue los siguientes puntos:

- Inicialmente, se genera el número de nodos del grafo que se calcula según los parámetros de entrada de la siguiente forma:

$$N = m^K \tag{5.1}$$

  donde *m* representa la dimensión o número de filas o columnas (m=n) de la matriz generadora θ.

- A continuación, los nodos se vuelcan en un fichero (o varios ficheros, dependiendo del número de *mappers* establecido) como una lista de identificadores, uno por línea.

- Por otro lado, se genera el número de aristas que tendrá el grafo, el cual viene dado por la siguiente fórmula:

$$E = (\sum_{i,j} \theta[i,j])^K \tag{5.2}$$

- El número de aristas se divide entre el número de *mappers* especificado como parámetro para permitir así su cálculo en diferentes nodos del clúster Hadoop. El cálculo de los vértices viene dado por los siguientes pasos:

- La elección de qué nodos del grafo están conectados por una arista viene dada por la siguiente función:

$$\prod_{i=0}^{K-1} \theta[\left\lfloor \frac{u}{n^i} \right\rfloor n, \left\lfloor \frac{v}{n^i} \right\rfloor n] \tag{5.3}$$

  Esta expresión matemática puede entenderse como una recursividad descendente sobre la matriz de adyacencia del grafo a generar, escogiendo bloques de la matriz cada vez más pequeños hasta llegar a una celda concreta.

- Debido a que la generación de aristas se realiza en diferentes nodos del clúster, puede darse el caso de aristas repetidas. Este problema se resuelve fácilmente gracias a que el paradigma MapReduce agrupa por clave los datos generados por los *mappers* antes de ser procesados por los *reducers*, evitando así escribir en disco aristas repetidas.

RGen integra tres matrices generadoras extraídas de BDGS, las cuales se han obtenido a partir del modelado de tres grafos reales:

- Amazon Movie Reviews: 7.9 millones de valoraciones de 253,059 usuarios sobre 889,176 películas.
- Google Web Graph: 875,713 nodos representando páginas web y 5.1 millones de aristas representado los enlaces entre ellas.
- Facebook Social Graph: 4039 nodos simbolizando usuarios y 882,234 relaciones de amistad entre ellos.

A la hora de seleccionar un modelo, es posible introducir como parámetro la representación lineal de una matriz generadora modelada de cualquier otro grafo, o utilizar uno de los tres modelos disponibles. Para seleccionar el modelo es necesario especificar como parámetro uno de los siguientes valores: “amazon”, “google” o “facebook”.

Capítulo 6

Diseño y desarrollo

En este capítulo se describen las distintas etapas llevadas a cabo durante el diseño y desarrollo de RGen. La metodología ágil seguida en todo este proceso ha sido Scrum, como se ha descrito en el Capítulo 4. De forma resumida, durante cada Sprint de desarrollo se abordó una épica, es decir, un conjunto de historias de usuario (funcionalidades que aportan valor al usuario) agrupadas por su misma naturaleza, obteniendo al final de cada etapa un producto entregable. La duración de los Sprints ha sido variable, dependiendo de la complejidad del trabajo a realizar en cada uno de ellos.

A continuación, se presentan para cada uno de los cuatro Sprints del proyecto todas las tareas realizadas así como su estimación de tiempo y coste realizada al principio del mismo. Finalmente, se muestra un resumen de las estimaciones de coste y tiempo para el global del proyecto (Sección 6.6).

6.1 Preparación y estudio del dominio

Previamente al diseño y desarrollo de RGen, se dedicó un periodo de tiempo de aproximadamente un mes a estudiar el entorno y estado del arte de este tipo de herramientas de generación de datos. Una vez establecidos los principales objetivos que se querían alcanzar con el desarrollo de este TFG, se inició una etapa de familiarización con las tecnologías concretas que iban a utilizarse debido a la naturaleza de estos requisitos. Durante esta etapa, por lo tanto, se realiza una primera toma de contacto con Apache Hadoop, su funcionamiento interno y características principales, así como con el paradigma de programación y el sistema de ficheros distribuido en los que se apoya, MapReduce y HDFS, respectivamente. Se recopila también información acerca de las distintas soluciones utilizadas actualmente en la generación de datos, así como de los términos y definiciones más comunes en este contexto. Todo este conocimiento sirve para esbozar el camino a seguir a lo largo del desarrollo del proyecto. El resultado de todo este proceso de estudio del dominio y asimilación de conceptos se refleja

Benchmark	Generador utilizado
Testdfsio	-
WordCount	RandomTextWriter (Hadoop)
Sort	RandomTextWriter (Hadoop)
Grep	RandomTextWriter (Hadoop)
TeraSort	TeraGen (Hadoop)
PageRank	DataGen (HiBench)
Connected Components	DataGen (HiBench)
Bayes	DataGen (HiBench)
KMeans	GenKMeansDataset (Mahout)
Aggregation	DataGen (HiBench)
Join	DataGen (HiBench)
Scan	DataGen (HiBench)
Command	Definido por el usuario

Tabla 6.1: Cargas de trabajo y generadores de datos utilizados en BDEv

en los primeros capítulos de esta memoria, especialmente en los Capítulos 2 y 3.

6.2 Primer Sprint - Análisis del generador de HiBench

Como primer paso en el desarrollo de RGen, se analizaron las soluciones de generación de datos utilizadas por la herramienta de benchmarking BDEv [2, 3], de la cual partió la necesidad de este proyecto. Como se puede observar en la Tabla 6.1, buena parte de las cargas de trabajo disponibles en BDEv se nutren del generador interno (DataGen) integrado en otra suite de benchmarking (HiBench), además de usar otras soluciones de terceros como Hadoop y Mahout. Por lo tanto, se toma HiBench como base de RGen clonando el repositorio en el que se aloja [1]. A partir de aquí, se toma la última versión estable del repositorio de HiBench sobre la que trabajar, en nuestro caso la 7.0.

El siguiente paso es aislar el generador de datos en concreto (DataGen) del resto de funcionalidades de HiBench, el cual se encuentra dentro del proyecto general en el directorio *autogen*. Se elimina, por tanto, el resto de ficheros de HiBench que son innecesarios durante la generación de datos. Tenemos de esta forma un proyecto Maven principal, representado por el fichero *pom.xml* y ubicado en la raíz del repositorio de HiBench, que contiene la información y las dependencias necesarias para toda la suite. Además, dentro del directorio *autogen* se encuentra el sub-proyecto Maven correspondiente al generador de datos de HiBench, que será la base del desarrollo de RGen. El siguiente paso consiste en unificar todo el árbol de directorios con el que contamos, fusionando ambos ficheros *pom.xml*. Para ello es necesario añadir las dependencias necesarias para nuestro proyecto, que se encontraban previamente en

[1] https://github.com/Intel-bigdata/HiBench

el proyecto Maven principal, en el fichero *pom.xml* propio de RGen, eliminando además aquellas dependencias innecesarias. También se modifican otros parámetros globales del proyecto Maven, como son el nombre, artefacto, versión, etc.

A partir de este punto, se hace un estudio del diseño de DataGen mediante ingeniería inversa, y se realizan pruebas básicas que sirven como toma de contacto con la herramienta. Se busca entender la estructura de clases y la forma en que se generan los datos para poder facilitar después la labor de integración y desarrollo de nuevas funcionalidades. El resultado de este proceso es un diagrama Unified Modelling Language (UML) de clases, representado en la Figura 6.1, útil para no tener que consultar el código en etapas posteriores. La clase principal de entrada a la herramienta se denomina DataGen, en la cual, según las opciones especificadas y capturadas por la clase auxiliar DataOptions, se crea un objeto concreto dependiendo del tipo de dato a generar. Las cuatro opciones disponibles son las clases BayesData, HiveData, PageRankData, NutchData. Estas clases llevan el nombre de las cargas de trabajo a las que alimentan. Existe una clase separada de esta estructura, GenKMeansDataset, en otro paquete distinto dentro del proyecto, importada de la parte de clústering de la librería Mahout [13], la cual también se integrará dentro de RGen. La siguiente lista describe brevemente la naturaleza de los datos generados por cada una de las clases:

- **BayesData**: Genera documentos de texto cuyas palabras siguen una distribución Zipfian [16] utilizando el diccionario por defecto de Linux.

- **HiveData**: Produce como salida datos web con enlaces que siguen una distribución Zipfian, en tablas relacionales.

- **PageRankData**: Genera grafos representando datos Web cuyos enlaces siguen una distribución Zipfian.

- **NutchData**: Genera una representación en grafos y texto de datos web cuyos enlaces y palabras en cada página siguen una distribución Zipfian, utilizando el diccionario por defecto de Linux.

Las clases correspondientes a la generación de flujos de datos o streams han sido eliminadas del proyecto debido a que no son necesarias en BDEv. Se agrupan además las diversas clases del proyecto buscando mejorar la organización de la estructura en paquetes de forma que representen los tipos de datos a generar.

La planificación llevada a cabo en este primer Sprint, mostrando las tareas y su estimación económica, se presenta en la Figura 6.2.

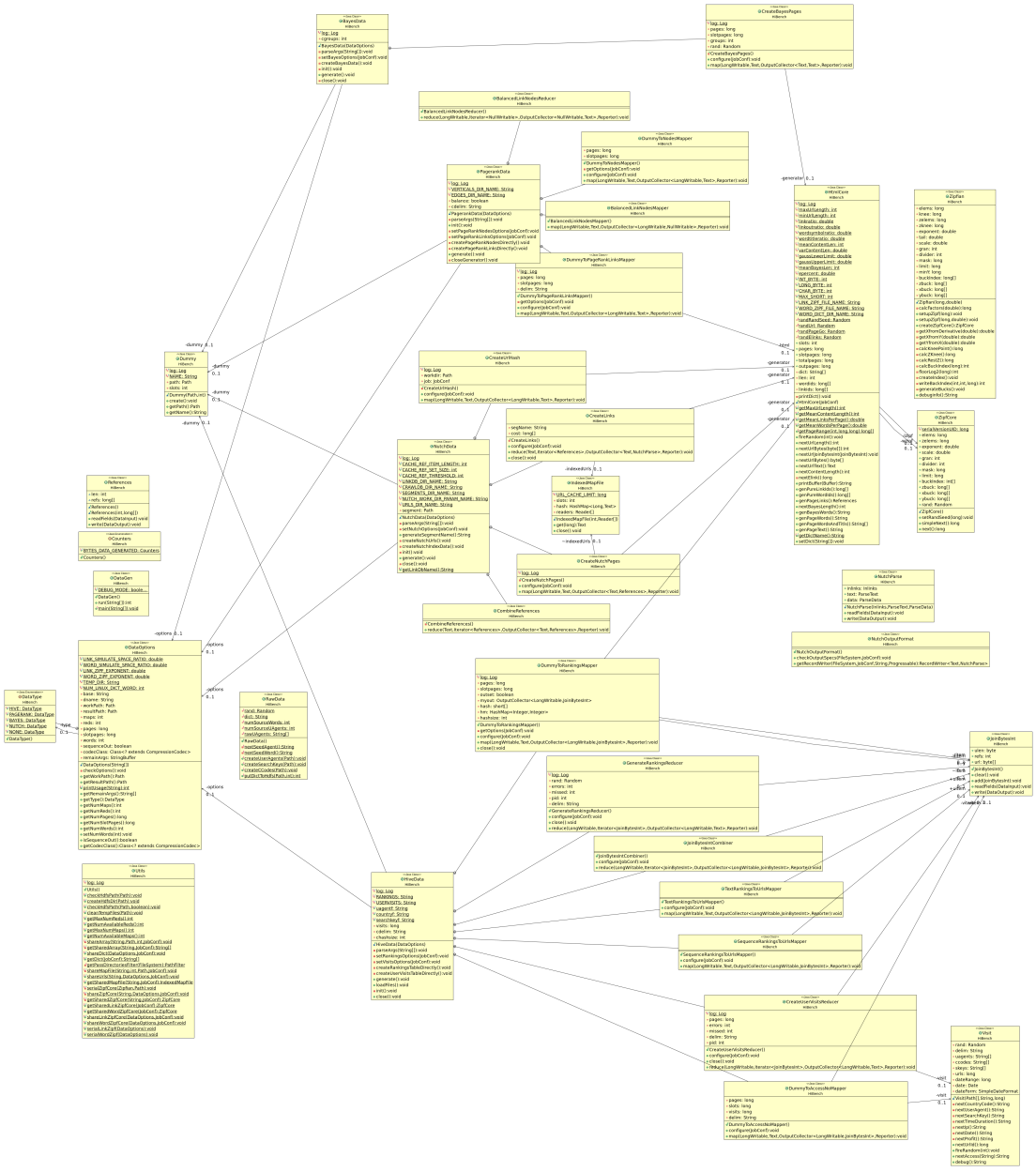


Figura 6.1: Diagrama UML del generador DataGen de HiBench

Sprint	Nombre de la tarea	Comienzo	Fin	Trabajo	Coste
1	Toma de contacto y pruebas herramienta base	03/02/2020	06/02/2020	16 horas	€ 400,00
1	Adaptación del proyecto	07/02/2020	11/02/2020	20 horas	€ 500,00
1	Diseño	12/02/2020	14/02/2020	12 horas	€ 300,00

Figura 6.2: Planificación y estimación del Sprint 1

6.3 Segundo Sprint - Integración

El siguiente paso en el desarrollo de RGen consiste en la integración de otras soluciones utilizadas en BDEv para la generación de datos. El objetivo fundamental de este segundo Sprint es la obtención de una herramienta totalmente independiente de HiBench y que cubra todas las necesidades de generación de datos como entrada de las cargas de trabajo disponibles en BDEv. Consultando de nuevo la Tabla 6.1 observamos que, además del generador interno de HiBench, BDEv utiliza tres soluciones adicionales:

- **RandomTextWriter**: Clase nativa del framework Hadoop que genera texto de forma aleatoria basado en un diccionario de datos preestablecido. Se utiliza en tres micro-benchmarks de BDEv: WordCount, Sort y Grep.
- **TeraGen**: Clase incluida en el paquete *terasort* de Hadoop encargada de generar el número de filas deseado para nutrir el popular benchmark estándar conocido como TeraSort.
- **GenKMeansDataset**: Clase proveniente del paquete *org.apache.mahout. clustering.kmeans* de Mahout que permite generar datos de entrada para el benchmark KMeans. Esta clase fue renombrada en RGen como KMeans.

Todo el proceso de análisis de las clases anteriores, con el objetivo de entender su estructura y funcionamiento, además de confeccionar los diagramas de clases UML ajustados para su integración en RGen (ver Figuras 6.3, 6.4, 6.5), conforman la parte de diseño en la planificación (primera tarea del Sprint).

En el caso de TeraGen y KMeans, estos generadores se integraron en RGen sin necesidad de realizar cambios significativos en su código, más allá de la adquisición de los argumentos de entrada para su correcto funcionamiento. Para el generador RandomTextWriter, sin embargo, a pesar de no modificar el algoritmo de generación que aborda la selección de palabras, sí que fue necesario cambiar la forma de adquirir los argumentos de entrada para hacerlo de la misma forma que en el resto de generadores. El objetivo es proporcionar en RGen una interfaz de uso uniforme al usuario final en la medida de lo posible. Por ejemplo, se eliminó la posibilidad de pasar parámetros a través de la configuración del trabajo de Hadoop, algo totalmente innecesario para nuestro caso de uso.

Más específicamente, los parámetros aceptados por RandomTextWriter serán únicamente tres: el número de bytes a generar, el número de tareas *map* a ejecutar y el directorio de salida. Para TeraGen los parámetros son muy parecidos, a excepción de los bytes generados, que en este caso son filas. En KMeans, sin embargo, existen otros parámetros específicos de este benchmark, como el número de clústeres, dimensión de la muestra, número de iteraciones del algoritmo, etc.

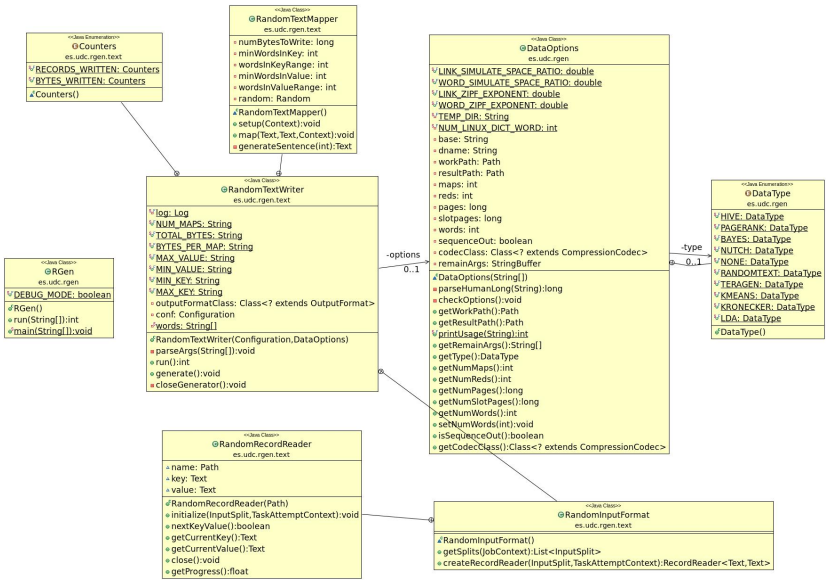


Figura 6.3: Diagrama UML del generador RandomTextWriter

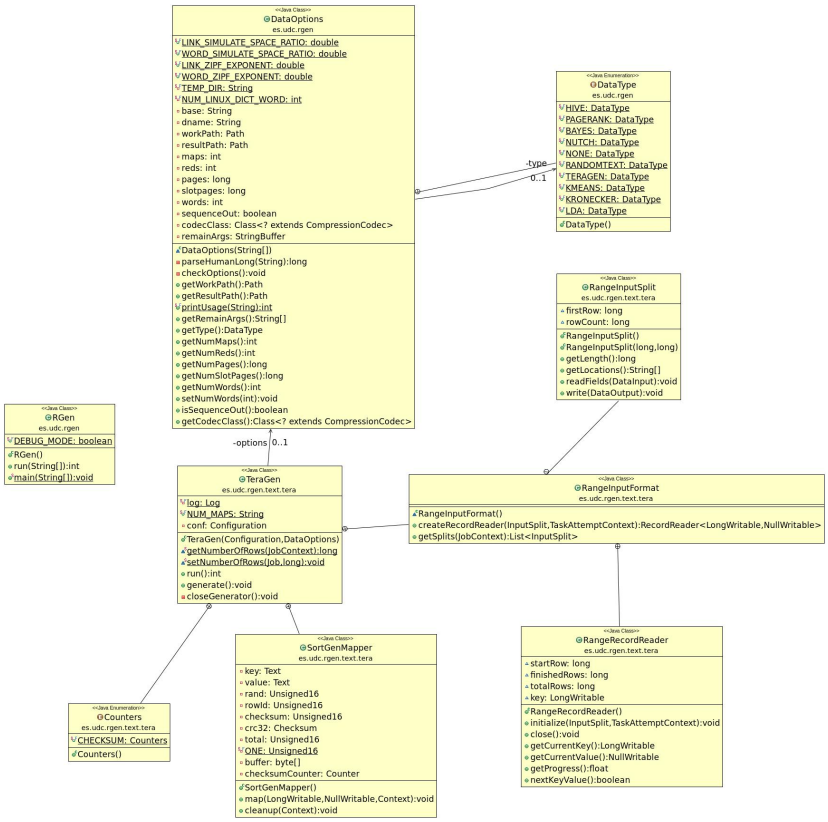


Figura 6.4: Diagrama UML del generador TeraGen

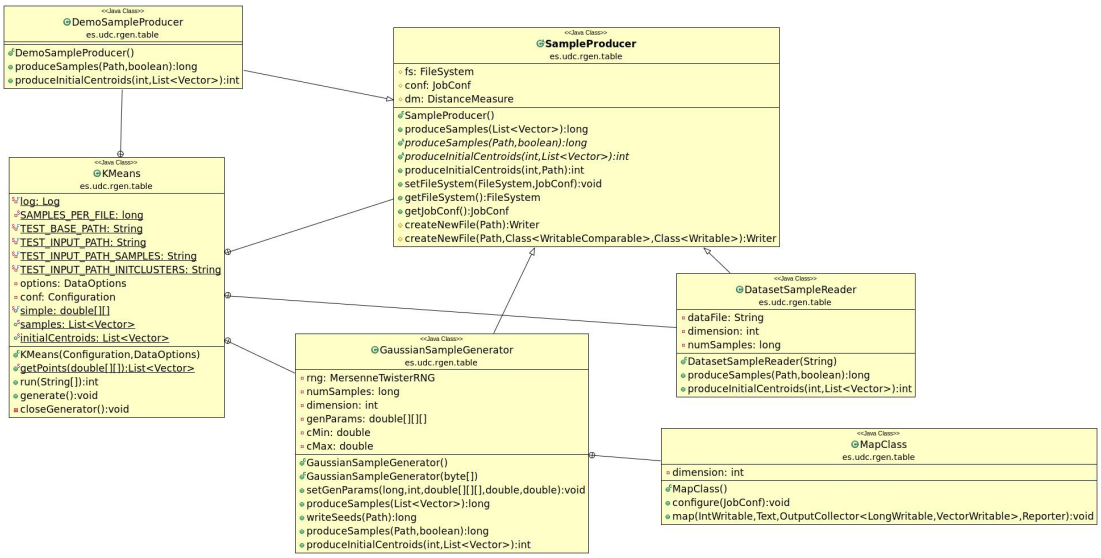


Figura 6.5: Diagrama UML del generador KMeans

Sprint	Nombre de la tarea	Comienzo	Fin	Trabajo	Coste
2	Diseño	17/02/2020	19/02/2020	12 horas	€ 300,00
2	Integración de RandomTextWritter	20/02/2020	28/02/2020	36 horas	€ 900,00
2	Integración de TeraGen	02/03/2020	06/03/2020	20 horas	€ 500,00
2	Integración de GenKMeansDataset	09/03/2020	13/03/2020	20 horas	€ 500,00

Figura 6.6: Planificación y estimación del Sprint 2

Cabe destacar también que la forma habitual utilizada para pasar información, como pueden ser parámetros necesarios para la ejecución de las tareas a los *mappers* y *reducers*, se realiza a través de la configuración del trabajo Hadoop a ejecutar en el clúster. Así, se define en cada clase una cadena de texto estática por cada parámetro a soportar, accesible desde cualquier parte del código, correspondiente a cada clave que guardará un valor de un tipo correspondiente. Una vez que los parámetros de entrada son parseados y procesados, se guardan por la clave deseada en la configuración del trabajo Hadoop para que así puedan ser accesibles desde todos los nodos cuando ejecutan una tarea *map* o *reduce*.

Finalmente, la planificación en tareas del segundo Sprint y su estimación económica se muestra en la Figura 6.6.

6.4 Tercer Sprint - Generación de texto mediante LDA

Una vez realizada la labor inicial de aislamiento e integración de las funcionalidades deseadas de otros generadores en nuestra herramienta, los siguientes dos Sprints se enfocan en desarrollar las nuevas funcionalidades proporcionadas por RGen. En concreto, el tercer Sprint

aborda el desarrollo de la generación de texto a partir del modelo LDA, descrito de forma teórica en la Sección 2.3.1 junto con la descripción de su funcionalidad en RGen en la Sección 5.1. En resumen, esta técnica se basa en generar documentos sintéticos que están formados a partir de un conjunto de palabras y siguiendo unos tópicos extraídos del análisis de un conjunto de documentos de entrada utilizando el modelo LDA.

El primer paso de este Sprint ha sido el estudio en profundidad del proceso de aprendizaje automático realizado por el modelo generativo LDA. Básicamente, este proceso permite que conjuntos de observaciones puedan ser explicados por grupos no observados, que indican por qué algunas partes de los datos son similares. Además, también se realiza un estudio de cómo aprovechar este conocimiento para generar texto que siga los tópicos hallados, lo cual fue implementado previamente por BDGS [11], pero de forma secuencial. Una vez analizado el algoritmo original implementado en BDGS, se realiza un diseño para su versión en paralelo siguiendo el paradigma MapReduce que preserve sus características principales.

El siguiente paso consiste en plasmar las ideas recogidas por el estudio anterior en un diagrama UML mostrado en la Figura 6.7, siguiendo la estructura de clases heredada y adaptada de Sprints anteriores que nos facilita la ampliación de funcionalidades en RGen. En este mapa visual se recogen los parámetros y procesos básicos durante todo el proceso de generación, desde el parseado y procesado de los parámetros de entrada necesarios hasta la ejecución del trabajo Hadoop. En este caso, debido a la naturaleza de esta funcionalidad, este trabajo solo contará con una función *map*, pues no es necesaria ninguna fase de agregación de los datos.

Una vez realizado el diseño comienza la etapa de desarrollo, la cual se divide en una serie de pasos que facilitan la organización y fidelidad al diseño. Primero se crea el esqueleto de clases mostrado previamente en el diagrama de la Figura 6.7. Después se implementa la adquisición y procesamiento de los parámetros de entrada que serán especificados por el usuario. El tercer paso consiste en el desarrollo del algoritmo paralelo que realizará la generación de los

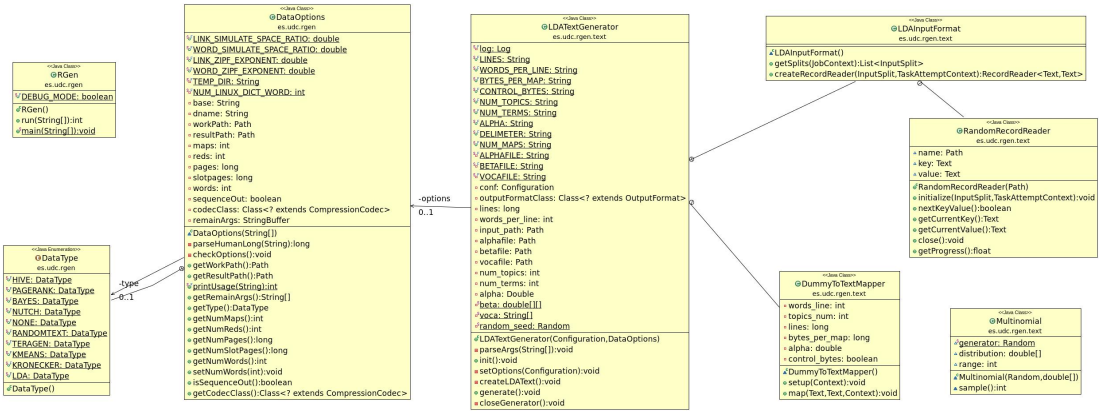


Figura 6.7: Diagrama UML del generador LDA

Sprint	Nombre de la tarea	Comienzo	Fin	Trabajo	Coste
3	Diseño	16/03/2020	20/03/2020	20 horas	€ 500,00
3	Creación estructura de clases	23/03/2020	27/03/2020	20 horas	€ 500,00
3	Implementación de extracción de parámetros	30/03/2020	03/04/2020	20 horas	€ 500,00
3	Implementación algoritmo de generación LDA	06/04/2020	17/04/2020	40 horas	€ 1.000,00
3	Prueba funcionalidad y correción errores	20/04/2020	24/04/2020	20 horas	€ 500,00
3	Aplicación de mejoras y posibles errores derivados	27/04/2020	01/05/2020	20 horas	€ 500,00

Figura 6.8: Planificación y estimación del Sprint 3

datos a partir de los parámetros establecidos. Cabe destacar que se ha implementado una clase Multinomial para dotarnos de las funcionalidades de este tipo de distribución, necesaria para el funcionamiento del algoritmo. Llegados a este punto tenemos la funcionalidad deseada, aunque con algunos errores y posibles mejoras, que son aplicadas posteriormente.

La planificación llevada a cabo durante el tercer Sprint, además de su estimación económica correspondiente, se muestra en la Figura 6.8.

6.5 Cuarto Sprint - Generación de grafos mediante Kronecker

La segunda funcionalidad que se añade a RGen es la generación de grafos a partir del modelo Kronecker. La estructura de pasos a seguir es, en general, muy similar a la de la sección anterior para el modelo LDA, la cual resulta útil en caso de querer añadir cualquier otra funcionalidad extra a nuestra herramienta en el futuro. El fundamento teórico del modelo Kronecker y cómo esta funcionalidad se proporciona específicamente en RGen se han descrito previamente en las Secciones 2.3.2 y 5.2, respectivamente. Resumiendo dichas secciones, la generación de grafos a partir del modelo Kronecker se basa en utilizar la potencia Kronecker de una matriz inicial para crear así un grafo con el número de nodos deseado y unas características similares a su ancestro.

Igual que en el Sprint de desarrollo anterior, el primer paso consiste en estudiar en detalle el proceso de aprendizaje a partir del cual obtenemos la matriz generadora con la que trabajaremos posteriormente. Se realiza además un estudio de cómo implementar en código el concepto de la potencia de Kronecker, anteriormente realizado en BDGS pero de forma secuencial, siguiendo para ello las directrices del paradigma MapReduce para obtener una versión paralela del mismo.

Una vez contamos con el conocimiento teórico necesario, plasmamos las ideas principales en un diagrama UML de clases que nos servirá de guía a la hora de implementarlo en el código fuente (ver Figura 6.9), adoptando una vez más la estructura de clases de las funcionalidades previamente implementadas. En este caso, para la generación de grafos sintéticos necesitaremos dos trabajos Hadoop. El primero se encargará de generar los nodos del grafo

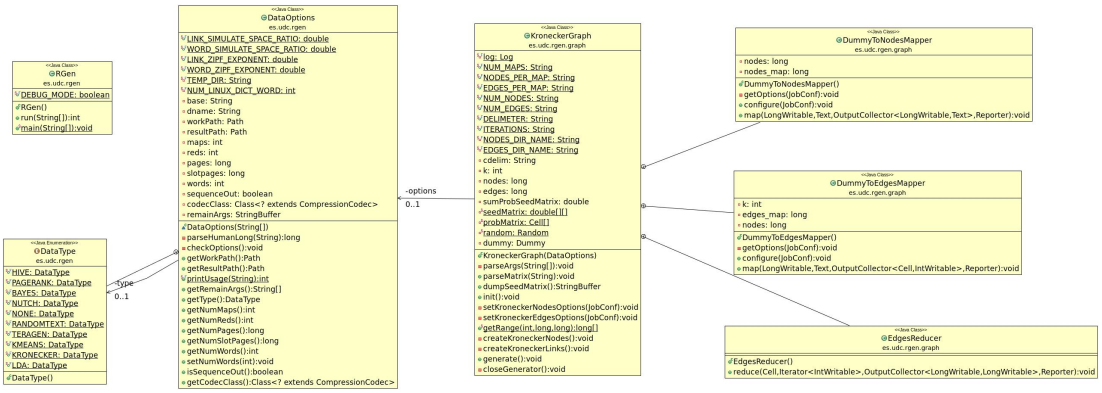


Figura 6.9: Diagrama UML del generador Kronecker

Sprint	Nombre de la tarea	Comienzo	Fin	Trabajo	Coste
4	Diseño	04/05/2020	08/05/2020	20 horas	€ 500,00
4	Creación estructura de clases	11/05/2020	15/05/2020	20 horas	€ 500,00
4	Implementación de extracción de parámetros	18/05/2020	22/05/2020	20 horas	€ 500,00
4	Implementación algoritmo de generación Kronecker	25/05/2020	05/06/2020	40 horas	€ 1.000,00
4	Prueba funcionalidad y correción errores	08/06/2020	12/06/2020	20 horas	€ 500,00
4	Aplicación de mejoras y posibles errores derivados	15/06/2020	19/06/2020	20 horas	€ 500,00

Figura 6.10: Planificación y estimación del Sprint 4

y ejecutará solamente la fase *map*. Con el segundo trabajo Hadoop generaremos el número de aristas calculado: una por cada iteración de un bucle que ejecutará cada tarea *map*. En este segundo trabajo contaremos también con una fase *reduce* que suprimirá las aristas repetidas y volcará los resultados finales en los ficheros de salida.

Sin embargo, además del algoritmo, se debe crear primero la estructura de clases necesarias para trabajar de forma más ordenada y la implementación de la adquisición de los parámetros de entrada para realizar posteriormente la generación del grafo. Una vez obtenemos la funcionalidad deseada se pasa a una fase de corrección de errores detectados en las pruebas y mejoras que van surgiendo.

La planificación que se ha llevado a cabo en el último Sprint, además de la estimación económica, se muestran en la Figura 6.10.

6.6 Estimaciones y coste

A lo largo de las secciones anteriores donde se describe el trabajo realizado en cada Sprint, también se muestra la planificación y la estimación económica detallada de cada uno. En la Figura 6.11 se muestra de forma global la información expuesta en los apartados anteriores correspondiente al desarrollo del proyecto. El periodo de desarrollo transcurre a lo largo de

Sprint	Comienzo	Fin	Estimado	Trabajo	Coste
Sprint 1	03/02/2020	14/02/2020	60 horas	48 horas	€ 1.200,00
Sprint 2	17/02/2020	13/03/2020	92 horas	88 horas	€ 2.200,00
Sprint 3	16/03/2020	01/05/2020	130 horas	140 horas	€ 3.500,00
Sprint 4	04/05/2020	19/06/2020	130 horas	140 horas	€ 3.500,00
			Total	416 horas	€ 10.400,00

Figura 6.11: Resumen de la planificación y estimación económica del desarrollo de RGen

Ámbito	Trabajo	Coste
Estudio del entorno y preparación	70 horas	1.750,00 €
Desarrollo	416 horas	10.400,00 €
Realización y análisis de mediciones	40 horas	1.000,00 €
Escritura de la memoria	100 horas	2.500,00 €
Tutores	50 horas	2.500,00 €
Total	676 horas	18.150,00 €

Figura 6.12: Resumen del coste y esfuerzo globales del proyecto

aproximadamente 4 meses y medio, comenzando en Marzo y terminando en Junio, dedicando una media de 20 horas semanales, lo que se resume en un total de 416 horas empleadas.

La parte de desarrollo, sin embargo, no representa la totalidad del TFG. En la Figura 6.12 se puede observar la planificación y la estimación económica de todo proyecto en su conjunto. Además de la parte de desarrollo expuesta anteriormente, se ha necesitado una fase inicial para el estudio y preparación del entorno de aproximadamente un mes de duración, como se explicó en la Sección 6.1. Una vez terminada la herramienta, se ha realizado su evaluación experimental en un entorno clúster real, además del tiempo necesario para la redacción y corrección del presente documento. En la Figura 6.13 se muestra en forma de diagrama de Gantt la planificación del proyecto a lo largo de todo el periodo de ejecución del mismo.

En total, el tiempo empleado en la realización del TFG asciende a 7 meses aproximadamente, desde Febrero hasta Agosto. Se ha tenido en cuenta también las horas empleadas por los tutores del proyecto en labores de dirección, reuniones de seguimiento, ayuda y correcciones, con un total de unas 50 horas. Para el cálculo del coste total del proyecto, se estima el salario de un desarrollador *freelancer* de 25 €/hora, y un salario estimado para los tutores de 50 €/hora.

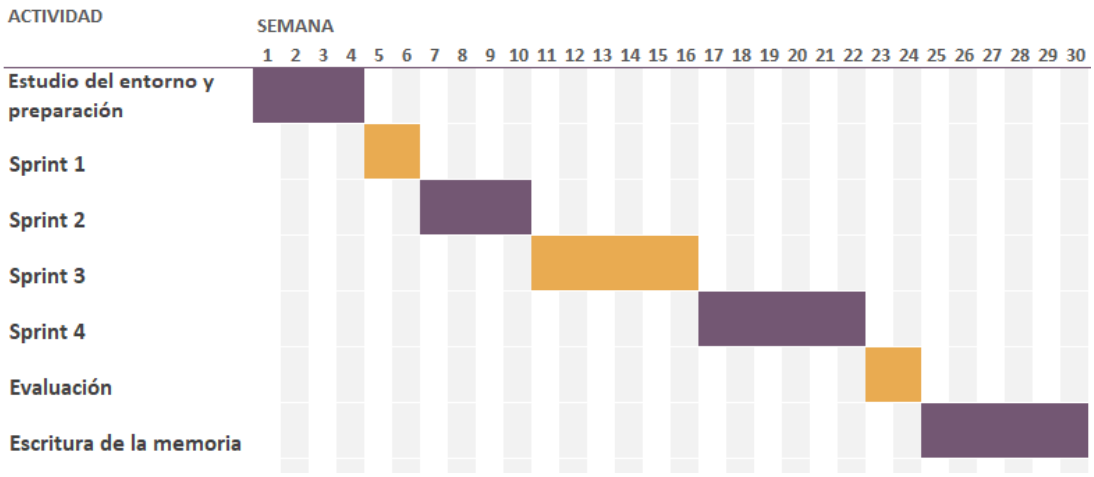


Figura 6.13: Diagrama de Gantt del proyecto

Capítulo 7

Evaluación del rendimiento

En este capítulo se exponen las pruebas de evaluación realizadas en un clúster de alto rendimiento. La finalidad principal de esta evaluación experimental de RGen no es tanto analizar la eficacia o velocidad específica a la que se genera un conjunto de datos concreto, sino evaluar su escalabilidad variando el volumen de datos a generar y la cantidad de recursos hardware utilizados. La evaluación se realiza sobre las dos funcionalidades que se han creado desde cero en este proyecto, la generación de texto LDA y de grafos Kronecker, y se toma como referencia otras dos funcionalidades integradas en RGen que generan el mismo tipo de datos.

7.1 Entorno de pruebas

El entorno en el cual se han realizado todas las pruebas experimentales, tanto de funcionamiento como de eficiencia, ha sido el clúster heterogéneo Pluton [25], administrado por el Grupo de Arquitectura de Computadores (GAC) de la Universidade da Coruña y alojado actualmente en el CPD del CITIC (https://www.citic-research.org).

En la actualidad, el clúster cuenta con los siguientes recursos hardware:

- Un nodo frontal que sirve como punto único de acceso a los usuarios para la interacción con el clúster.

- 23 nodos de cómputo que se encargan de proporcionar los recursos computacionales para la ejecución de las aplicaciones. En conjunto, la capacidad de cómputo agregada es de 432 núcleos o cores físicos (864 cores lógicos), con un total de 2.2 TiB de memoria principal. Los nodos están interconectados por dos redes: Gigabit Ethernet e InfiniBand FDR, siendo esta última una red de baja latencia y alto ancho de banda. Algunos nodos disponen además de aceleradores (e.g. GPUs), que no se han utilizado para nuestros experimentos.

Entre los distintos tipos de nodos de cómputo que hay disponibles en Pluton, para la evaluación de RGen se han usado nodos que cuentan con 16 cores físicos (32 cores lógicos), 64 GB de memoria y un disco duro local de 1 TB para ser usado con HDFS. Esta información es importante tenerla en cuenta a la hora de configurar las pruebas. Además, la red de interconexión entre estos nodos es Infiniband FDR, proporcionando hasta 56 Gbps de ancho de banda.

En lo que al software se refiere, el clúster cuenta con el entorno de administración Rocks 7, una distribución GNU/Linux basada en CentOS 7, la cual es libre y soportada por la comunidad. Además, los trabajos de los usuarios se administran mediante el gestor Slurm Workload Manager. Entre el software disponible en el clúster, en concreto para este proyecto se ha hecho uso de la máquina virtual de Java OpenJDK v1.8.0_262, así como del código de BDEv v3.4 y Hadoop v2.10 necesarios para la ejecución de las pruebas.

7.2 Diseño de las pruebas

La finalidad de las pruebas realizadas con RGen es demostrar la escalabilidad de la herramienta a la hora de generar datos en paralelo en un entorno distribuido. No era objetivo de este proyecto minimizar el tiempo necesario para generar un determinado conjunto de datos, ya que eso depende en gran medida de la cantidad y complejidad del cómputo que haya que realizar en función del tipo de datos a generar y su naturaleza. Lo que se pretende analizar con las pruebas es el comportamiento de RGen cuando se varía el volumen de datos a generar y los recursos hardware utilizados. Las pruebas realizadas evalúan las dos nuevas funcionalidades que proporciona RGen: la generación de texto a partir del modelo LDA y la generación de grafos mediante el modelo Kronecker. Adicionalmente, se realizaron las mismas pruebas usando dos generadores integrados en RGen que generan el mismo tipo de datos: texto aleatorio (RandomTextWriter) y grafos (PageRank), tal como se describió en las Secciones 6.3 y 6.3, respectivamente. Estos resultados se toman como referencia para hacer un análisis comparativo.

La escalabilidad, en general, se puede definir como la capacidad de una aplicación paralela (o algoritmo) de mantener sus prestaciones cuando se aumenta el número de recursos computacionales (e.g. cores) y/o el tamaño del problema a resolver. Es decir, la escalabilidad indica la capacidad que tiene una aplicación paralela de utilizar de forma eficiente un incremento de los recursos computacionales. Se puede distinguir entre escalabilidad débil y escalabilidad fuerte. En la escalabilidad débil, se aumenta el número de cores manteniendo constante la carga de trabajo para cada uno de ellos. El objetivo es poder tratar un tamaño de problema más grande en un tiempo de ejecución similar incrementando para ello los recursos de forma proporcional. En nuestro contexto, se analizó la capacidad que tiene RGen de mantener un tiempo de ejecución similar cuando se aumenta el número de nodos de cómputo que se utilizan durante

#Nodos de cómputo	Volumen de datos	
	Escalabilidad débil	Escalabilidad fuerte
2	40 GB	320 GB
4	80 GB	320 GB
8	160 GB	320 GB
16	320 GB	320 GB

Tabla 7.1: Pruebas de generación de texto

#Nodos de cómputo	Volumen de datos	
	Escalabilidad débil	Escalabilidad fuerte
2	67 M nodos	536 M nodos
4	134 M nodos	536 M nodos
8	268 M nodos	536 M nodos
16	536 M nodos	536 M nodos

Tabla 7.2: Pruebas de generación de grafos

la generación, manteniendo constante el volumen de datos a generar por cada nodo (e.g. 20 GB para generación de texto). Por otro lado, en la escalabilidad fuerte, se mantiene constante el tamaño del problema. El objetivo es minimizar el tiempo de ejecución aumentando el número de cores, ya que así la carga de trabajo por core disminuye. En nuestro caso, se analizó la capacidad de RGen de disminuir el tiempo de ejecución para generar un conjunto de datos de un determinado tamaño fijo aumentando el número de nodos de cómputo utilizados.

Las Tablas 7.1 y 7.2 muestran un resumen de todas las pruebas realizadas, utilizando diferentes configuraciones del clúster (i.e. número de nodos), para la generación de texto y grafos, respectivamente. El volumen de datos a generar se expresa en GB para el texto y en millones de nodos para los grafos. Cabe destacar que, como se ha comentado en la sección anterior, cada nodo de cómputo cuenta con 16 cores físicos, por lo que se ha configurado Hadoop para ejecutar 8 *mappers* y 8 *reducers* por nodo. Para la generación de texto LDA se ha utilizado el modelo *lda_wiki1w* y para la generación de grafos Kronecker el modelo *google*, ambos integrados en RGen.

Para facilitar el lanzamiento de los trabajos en el clúster se ha utilizado BDEv, ya que nos permite desplegar Hadoop en tantos nodos como se soliciten al gestor Slurm de Pluton, ejecutar el experimento especificado generando los datos sobre HDFS, obtener todo tipo de estadísticas sobre la ejecución y, por último, borrar los datos generados innecesarios para nuestro estudio.

		Número de nodos			
		2	4	8	16
Escalabilidad débil	Aleatorio	143	178	205	228
	LDA	1150	1140	1142	1165
Escalabilidad fuerte	Aleatorio	1800	1085	445	228
	LDA	8843	4457	2263	1165

Tabla 7.3: Tiempos de ejecución para la generación de texto (en segundos)

		Número de nodos			
		2	4	8	16
Escalabilidad débil	PageRank	250	289	274	316
	Kronecker	343	380	414	553
Escalabilidad fuerte	PageRank	2739	1596	709	316
	Kronecker	7823	2426	848	553

Tabla 7.4: Tiempos de ejecución para la generación de grafos (en segundos)

7.3 Análisis de los resultados

Una vez ejecutados todos los experimentos correspondientes a las pruebas descritas en la sección anterior, se realiza un proceso de recolección y análisis de los resultados obtenidos. Los datos correspondientes a las pruebas de generación de texto se muestran en la Tabla 7.3, mientras que los datos relativos a las pruebas de generación de grafos se muestran en la Tabla 7.4. Todos los tiempos están expresados en segundos. Lo primero que se puede observar en una primera aproximación es que tanto la generación de texto LDA como de grafos Kronecker tardan más tiempo en ejecutarse que sus análogas usando los mismos recursos. Este comportamiento era esperable debido a que la naturaleza y carga computacional de ambos algoritmos de generación es muy distinta. Por ejemplo, la generación de texto de forma aleatoria no implica prácticamente ningún tipo de cómputo significativo durante la ejecución de las tareas *map*, por lo que es perfectamente lógico obtener tiempos de ejecución significativamente menores que usando un modelo mucho más complejo como LDA.

Entrando más en profundidad en los resultados obtenidos, y representando los datos en gráficas para facilitar su interpretación visual, se puede afirmar que se han cumplido los objetivos perseguidos tanto para la escalabilidad débil como para la fuerte. En las Figuras 7.1 y 7.2 se muestran los resultados referentes a las pruebas de generación de texto para la escalabilidad débil y fuerte, respectivamente. Se aprecia en la Figura 7.1 que tanto la generación de texto aleatorio como mediante el modelo LDA presentan unos resultados de escalabilidad débil prácticamente constantes, lo que indica que ambas opciones son capaces de generar volúmenes de datos progresivamente ascendentes en tiempos de ejecución similares si se aumentan de forma proporcional los recursos computacionales (i.e. nodos de cómputo). Incluso

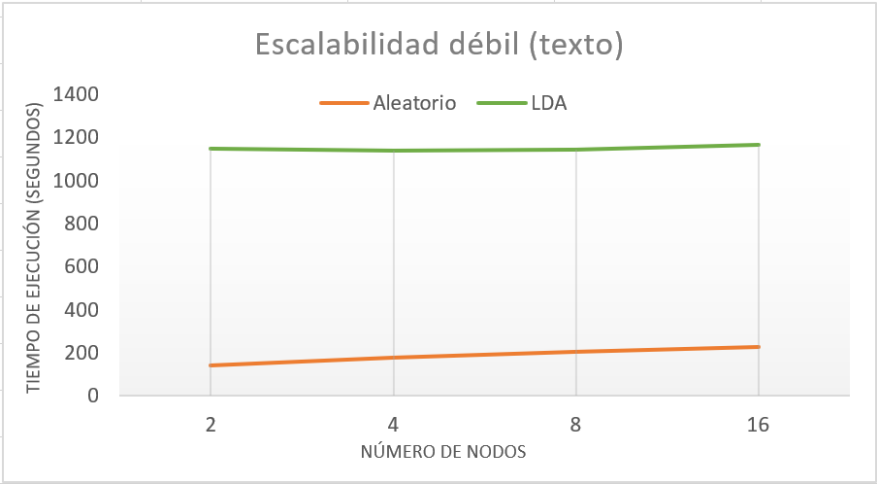


Figura 7.1: Escalabilidad débil (generación de texto)

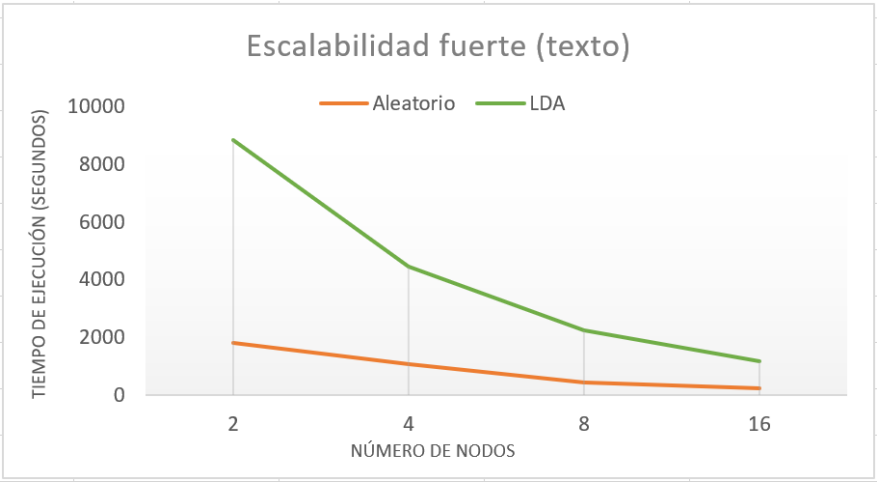


Figura 7.2: Escalabilidad fuerte (generación de texto)

es posible afirmar que la generación mediante LDA obtiene resultados ligeramente mejores que la generación aleatoria, la cual muestra una pequeña subida en sus tiempos usando 8 y 16 nodos. Por otra parte, se observa en la Figura 7.2 que ambos generadores presentan una buena escalabilidad fuerte, en el sentido de que si aumentamos el número de recursos utilizados para la generación de un volumen de datos específico, en este caso 320 GB, el tiempo de ejecución disminuye proporcionalmente. Estos resultados demuestran la escalabilidad casi lineal que proporciona el modelo MapReduce combinado con HDFS en casos de uso donde solo se ejecuta la fase *Map*, como ocurre al generar texto de forma aleatoria o mediante el modelo LDA.

De la misma forma se procede con el análisis de los resultados para la generación de grafos

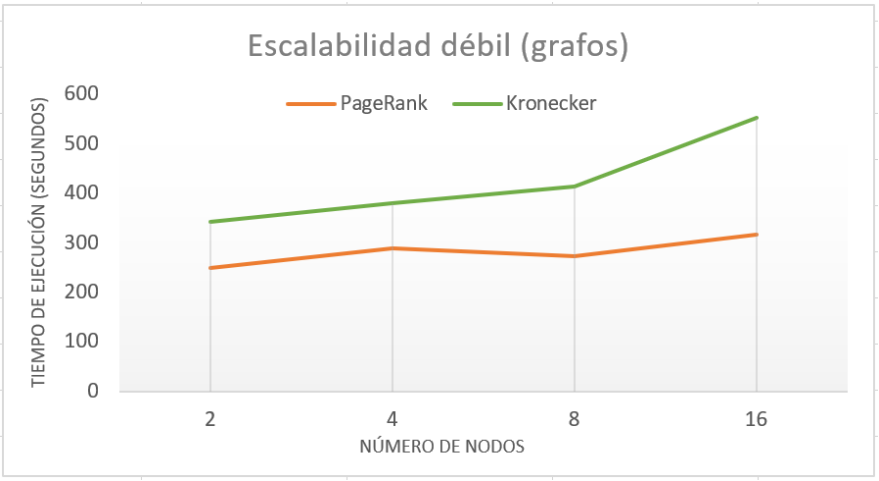


Figura 7.3: Escalabilidad débil (generación de grafos)

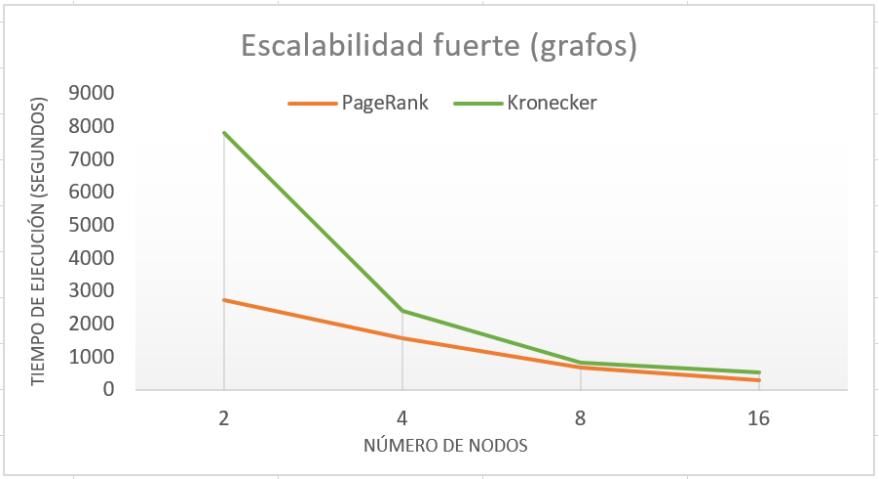


Figura 7.4: Escalabilidad fuerte (generación de grafos)

(ver Figuras 7.3 y 7.4). Se puede observar que tanto la generación de grafos PageRank como mediante el modelo Kronecker presentan una escalabilidad débil más irregular que en el caso anterior. De hecho, se aprecia una pequeña desviación para el modelo Kronecker cuando se usan 16 nodos de cómputo. Este comportamiento se debe principalmente a que este algoritmo ejecuta dos trabajos Hadoop (ver Sección 6.5) en lugar de uno, como ocurre en el generador de texto LDA, además de contar con la fase *Reduce*. Esto implica que la red de interconexión juegue un rol determinante en el rendimiento, puesto que es necesario el intercambio de datos entre los nodos del clúster durante la fase intermedia entre *Map* y *Reduce* (i.e. *Shuffle and Sort*). Se puede concluir que, a medida que aumenta el número de nodos de cómputo a utilizar, y por tanto aumenta el volumen de datos a generar y a intercambiar durante la operación de movimientos de datos (i.e. *shuffling*), la sobrecarga en la red se hace más evidente. En la Figura 7.4

observamos los resultados de la escalabilidad fuerte, donde es posible apreciar cómo el modelo Kronecker escala incluso mejor que el generador PageRank, cerrando significativamente el gap de rendimiento existente entre ambos cuando se usan más de 4 nodos de cómputo.

Después de analizar todos los resultados obtenidos, podemos concluir que los objetivos marcados de escalabilidad se han cumplido para las dos funcionalidades diseñadas y desarrolladas desde cero en RGen.

Capítulo 8

Conclusiones y trabajo futuro

EL objetivo principal de este proyecto ha consistido en el diseño y desarrollo de una herramienta paralela para la generación de datos basada en el paradigma de programación MapReduce y con soporte para el sistema de ficheros distribuido HDFS. Concebido en primera instancia como un generador a medida para las necesidades de la herramienta de benchmarking BDEv, RGen unifica, integra y amplía las soluciones utilizadas actualmente en BDEv soportando las características definidas por las 4 Vs de Big Data: Volumen, Velocidad, Variedad y Veracidad. A continuación se exponen las conclusiones extraídas tras analizar los resultados del trabajo realizado, el grado de consecución de los objetivos inicialmente marcados, y las posibles líneas futuras a seguir en caso de desear expandir las funcionalidades de RGen.

8.1 Conclusiones

Este proyecto se inició con una labor principalmente de integración en una única solución independiente que diese respuesta a la problemática de la generación de datos para cargas de trabajo Big Data en entornos distribuidos. Para ello se han especificado unos objetivos a cumplir que vienen dados en gran medida para satisfacer las necesidades de la herramienta BDEv. Es por ello que se aúnan en RGen las funcionalidades de las diversas herramientas que hasta el momento BDEv estaba utilizando en la fase de generación de los conjuntos de datos de entrada para las cargas de trabajo soportadas. Las características principales que presentan todos los generadores de datos en RGen son el uso del paradigma paralelo MapReduce y el almacenamiento de los datos directamente sobre el sistema de ficheros distribuido HDFS, lo cual elimina la necesidad de producir datos en local y copiarlos posteriormente a HDFS. Ambas características también aplican en el desarrollo de las dos nuevas funcionalidades añadidas en RGen (generación de texto con LDA y de grafos Kronecker), además de cumplir los requisitos de las 4 Vs en la generación de datos. Con unos resultados favorables, tal y como muestran los experimentos presentados en el Capítulo 7, podemos afirmar que se han conseguido los

objetivos marcados en el inicio del proyecto.

A nivel personal la realización de este proyecto me ha permitido iniciarme en dos grandes ámbitos apenas tratados durante mis estudios de Grado, el procesamiento Big Data y la generación de datos. Ambos tópicos me han parecido sumamente interesantes, especialmente el procesamiento Big Data debido a la importancia que está cobrando en la actualidad. En un primer momento puede resultar difícil emprender el camino del autoaprendizaje en cualquier rama de estudio, pero a la larga ayuda a mejorar la capacidad de resolución de problemas y a abrir horizontes de una manera autónoma. Sin duda seguiré estudiando y profundizando sobre todo lo aprendido a lo largo de mi TFG. Por otra parte, este proyecto me ha servido para afianzar los conocimientos y habilidades adquiridos durante estos cuatro años del Grado en Ingeniería Informática, como la aplicación práctica de una metodología ágil como Scrum, el uso de herramientas como Git y Maven que facilitan el control de versiones y dependencias de un proyecto real, etc. Por último, me gustaría destacar también lo enriquecedor que ha sido emplear un clúster de alto rendimiento para la realización de las pruebas en un entorno real, actividad que tampoco había llevado a cabo anteriormente.

Finalmente, cabe destacar también que la herramienta RGen se encuentra disponible para su descarga en el siguiente repositorio público de GitHub: `https://github.com/rubenperez98/RGen`. Todo el código heredado a partir del cual se construye una parte de esta herramienta es *open source*, con lo que se mantienen las licencias correspondientes, abriendo la posibilidad a cualquier usuario de añadir, modificar o mejorar cualquiera de las funcionalidades soportadas actualmente.

8.2 Trabajo futuro

Como se ha comentado en varias ocasiones a lo largo del presente documento, una de las ideas principales de este TFG es la integración de múltiples soluciones en una herramienta independiente. Es por ello que desde un primer momento se ha diseñado RGen con la idea de facilitar la ampliación de sus funcionalidades en un futuro, creando una aplicación fácilmente escalable que permita la integración de nuevas características sin necesidad de manipular y/o modificar las existentes. En este contexto, cualquier usuario podría hacer nuevos desarrollos en RGen para ampliar o mejorar su funcionalidades de una manera relativamente sencilla. Además, después del proceso de adaptación e integración llevado a cabo, resulta más fácil integrar cualquier característica nueva a la herramienta de forma directa y clara, gracias a la estructura troncal del código y a las buenas prácticas de diseño seguidas.

Una característica interesante que podría añadirse a RGen para ampliar su potencial es la generación de datos estructurados (i.e. tablas relacionales) a partir de la funcionalidad soportada por el Parallel Data Generation Framework (PDGF) [12]. PDGF es un generador de datos

flexible y genérico, capaz de generar grandes cantidades de datos relacionales muy rápido. Se creó inicialmente en la Universidad de Passau y se utiliza en el desarrollo de un benchmark industrial ETL (proceso de extracción, transformación y carga) estándar. Está basado en la generación paralela de números aleatorios para la producción independiente de valores relacionados. Además, no está implementado usando MapReduce ni soporta la generación de datos sobre HDFS, por lo que su integración en RGen siguiendo este paradigma sería de gran interés. Sería sin duda una de las opciones más plausibles a añadir en nuestra herramienta a corto plazo.

Otras mejoras posibles serían el soporte para la generación de otros tipos de datos (e.g. datos semi-estructurados como XML o JSON), la inclusión de parámetros de entrada adicionales a las funcionalidades existentes que añadan nuevas posibilidades al proceso de generación (e.g. la posibilidad de usar otras distribuciones matemáticas internamente), o la elaboración de estadísticas sobre el rendimiento de la herramienta después de cada ejecución.

Lista de acrónimos

BDEv Big Data Evaluator. 1, 2, 21

BDGS Big Data Generator Suite. 3

GAC Grupo de Arquitectura de Computadores. 21, 47

GFS Google File System. 11

HDFS Hadoop Distributed File System. 2, 18

IDE Integrated Development Environment. 23

IoT Internet of Things. 8

JVM Java Virtual Machine. 22

LDA Latent Dirichlet Allocation. 2, 12

PDGF Parallel Data Generation Framework. 3, 56

SNAP Stanford Network Analysis Platform. 16

TFG Trabajo Fin de Grado. 1

UML Unified Modelling Language. 37

YARN Yet Another Resource Negotiator. 17, 20

Glosario

4 Vs En el contexto de la generación de datos se refiere a las características que debería cumplir un generador: volumen, velocidad, variedad y veracidad. 2

BigDataBench Suite de benchmarking Big Data e IA escalable y unificada. 3

Hadoop Framework Big Data de código abierto especializado en el almacenamiento y procesamiento de grandes volúmenes de datos. 2

HiBench Suite de benchmarking Big Data que ayuda a evaluar diferentes frameworks de procesamiento distribuido en términos de velocidad, rendimiento y utilización de los recursos del sistema. 2

Kronecker Apellido del matemático alemán Leopold Kronecker, que da nombre a múltiples modelos y operaciones en el ámbito del análisis y generación de grafos. 2

Mahout Proyecto de la Apache Software Foundation para producir implementaciones gratuitas de algoritmos de aprendizaje automático distribuidos y escalables enfocados principalmente en álgebra lineal. 3

MapReduce Modelo de programación para dar soporte a la computación paralela sobre grandes colecciones de datos en clústeres de computadores. 2

Zipfian Distribución matemática que sigue la ley de Zipf, según la cual en una determinada lengua la frecuencia de aparición de distintas palabras sigue una distribución determinada. 9, 37

Bibliografía

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] J. Veiga, J. Enes, R. R. Expósito, and J. Touriño, "BDEv 3.0: Energy Efficiency and Microarchitectural Characterization of Big Data Processing Frameworks," *Future Generation Computer Systems*, vol. 86, pp. 565–581, 2018.

[3] "BDEv: Big Data Evaluator." [En línea]. Disponible en: http://bdev.des.udc.es

[4] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench Benchmark Suite: Characterization of the MapReduce-based Data Analysis," in *Proceedings of the IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. Long Beach, CA, USA, 2010, pp. 41–51.

[5] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST'2010)*. Incline Village, NV, USA, 2010, pp. 1–10.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[7] T. Ganegedara, "Intuitive Guide to Latent Dirichlet Allocation." [En línea]. Disponible en: https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158

[8] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker Graphs: An Approach to Modeling Networks," *Journal of Machine Learning Research*, vol. 11, no. 2, pp. 985–1042, 2010.

[9] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "Realistic, Mathematically Tractable Graph Generation and Evolution, Using Kronecker Multiplication," in *Pro-*

ceedings of the European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2005). Porto, Portugal, 2005, pp. 133–145.

[10] L. Wang *et al.*, "BigDataBench: A Big Data Benchmark Suite from Internet Services," in *Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture (HPCA'14)*. Orlando, FL, USA, 2014, pp. 488–499.

[11] Z. Ming, C. Luo, W. Gao, R. Han, Q. Yang, L. Wang, and J. Zhan, "BDGS: A Scalable Big Data Generator Suite in Big Data Benchmarking," in *Proceedings of the Workshop on Big Data Benchmarks (WBDB 2013)*. Xi'an, China, 2013, pp. 138–154.

[12] T. Rabl and H.-A. Jacobsen, "Big Data Generation," in *Proceedings of the Workshop on Big Data Benchmarks (WBDB 2012)*. Pune, India, 2012, pp. 20–27.

[13] "Distributed linear algebra framework and mathematically expressive Scala DSL." [En línea]. Disponible en: https://mahout.apache.org/

[14] R. Han, L. K. John, and J. Zhan, "Benchmarking Big Data Systems: A Review," *IEEE Transactions on Services Computing*, vol. 11, no. 3, pp. 580–597, 2017.

[15] M. J. Carey, "BDMS Performance Evaluation: Practices, Pitfalls, and Possibilities," in *Proceedings of the Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2012)*. Istanbul, Turkey, 2012, pp. 108–123.

[16] S. T. Piantadosi, "Zipf's Word Frequency Law in Natural Language: A Critical Review and Future Directions," *Psychonomic Bulletin & Review*, vol. 21, no. 5, pp. 1112–1130, 2014.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," *SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[18] K. Zhai, J. Boyd-Graber, N. Asadi, and M. L. Alkhouja, "Mr. LDA: A Flexible Large Scale Topic Modeling Package Using Variational Inference in MapReduce," in *Proceedings of the 21st World Wide Web Conference (WWW'12)*. Lyon, France, 2012, pp. 879–888.

[19] D. M. Blei, "LDA-C: A C Implementation of Latent Dirichlet Allocation (LDA)." [En línea]. Disponible en: https://github.com/blei-lab/lda-c

[20] J. Leskovec and R. Sosič, "SNAP: A General-Purpose Network Analysis and Graph-Mining Library," *ACM Transactions on Intelligent Systems and Technology*, vol. 8, no. 1, pp. 1–20, 2016.

[21] The Apache Software Foundation, "Apache Hadoop." [En línea]. Disponible en: https://hadoop.apache.org

[22] V. K. Vavilapalli *et al.*, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC'13)*. Santa Clara, CA, USA, 2013, pp. 5:1–5:16.

[23] The Apache Software Foundation, "Apache YARN." [En línea]. Disponible en: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[24] "The Home of Scrum." [En línea]. Disponible en: https://www.scrum.org

[25] "Clúster Pluton." [En línea]. Disponible en: http://pluton.dec.udc.es